



# Chapter 16 - Component-based software engineering

# Topics covered

---



- ✧ Components and component models
- ✧ CBSE processes
- ✧ Component composition

# Component-based development



- ✧ Component-based software engineering (CBSE) is an approach to software development that relies on the reuse of entities called 'software components'.
- ✧ It 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 stand-alone service providers. They can exist as stand-alone entities.

# CBSE essentials

---



- ✧ Independent components specified by their interfaces.
- ✧ Component standards to facilitate component integration.
- ✧ Middleware that provides support for component interoperability.
- ✧ 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:
  - Components are independent so do not interfere with each other;
  - Component implementations are hidden;
  - Communication is through well-defined interfaces;
  - One components can be replaced by another if its interface is maintained;
  - Component infrastructures offer a range of standard services.

# Component standards

---



- ✧ Standards need to be established so that components can communicate with each other and inter-operate.
- ✧ Unfortunately, several competing component standards were established:
  - Sun's Enterprise Java Beans
  - Microsoft's COM and .NET
  - CORBA's CCM
- ✧ In practice, these multiple standards have hindered the uptake of CBSE. It is impossible for components developed using different approaches to work together.

# Service-oriented software engineering

---



- ✧ An executable service is a type of independent component. It has a 'provides' interface but not a 'requires' interface.
- ✧ From the outset, services have been based around standards so there are no problems in communicating between services offered by different vendors.
- ✧ System performance may be slower with services but this approach is replacing CBSE in many systems.
- ✧ Covered in Chapter 18



# Components and component models



# Components



- ✧ Components provide a service without regard to where the component is executing or its programming language
  - A component is an independent executable entity that can be made up of one or more executable objects;
  - The component interface is published and all interactions are through the published interface;

# Mi a szoftver komponens?

- szoftver architektúra felőli megközelítés
- „Egy program vagy számítógépes rendszer szoftver architektúrája a rendszer struktúrája, ami tartalmazza
  - a szoftver komponenseket
  - a komponensek kívülről látható tulajdonságait, és
  - a komponensek közötti kapcsolatokat.”

*Bass, L., Clements P., Kazman R.: Software Architecture in Practice. Addison-Wesley, 2003.*

# Mi a szoftver komponens?

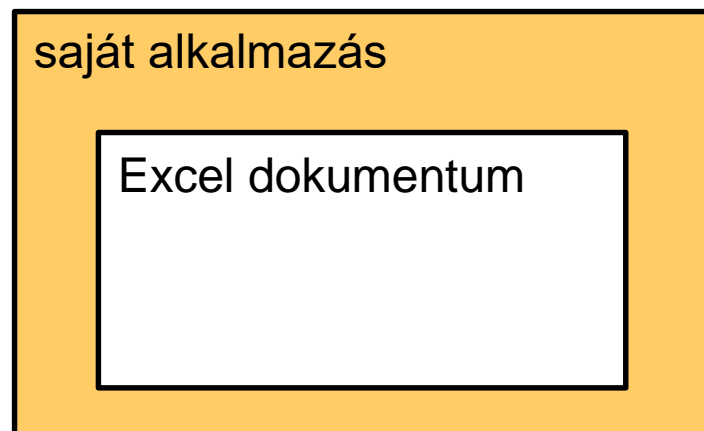
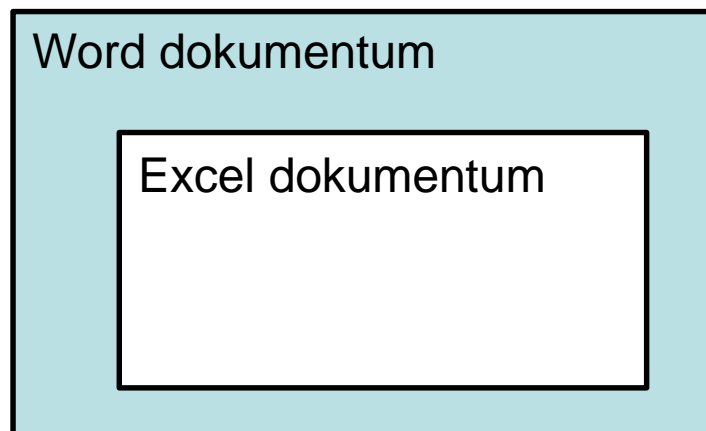
- korábbi sikeres komponensek:
- matematikai könyvtárak, matematikai és fizikai függvények, eljárások
- jellemzőik
  - jól definiált elmélet, szabványos
  - egyszerű interfész, eljárás hívás alapú kommunikáció a felhasználó és a komponens között
  - jól definiált bemenet és kimenet
  - viszonylag jó hibakezelés
  - nehezebb adaptálhatóság

# Mi a szoftver komponens?

- valamivel nagyobb méretű, kliens-szerver típusú komponensek
- adatbázisok
  - relációs adatbázisok, (objektum orientált, hierarchikus, stb.)
  - szabványos felület, SQL
- X-windows
  - szabványos felület, call-back típusú kommunikáció
  - magas szintű adaptálhatóság

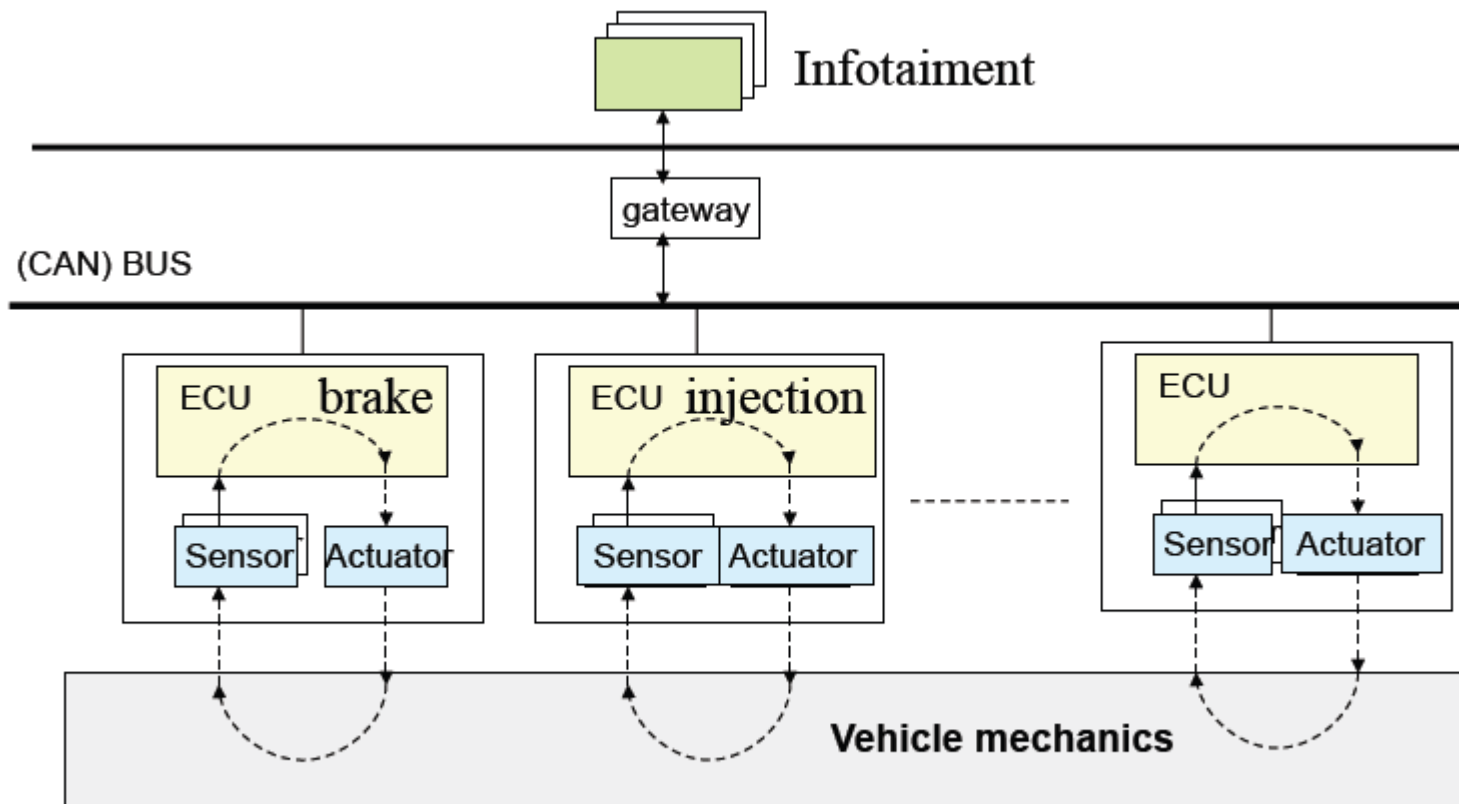
# Mi a szoftver komponens?

- „igazi” komponensek
  - Component Object Management – COM
  - Enterprise JavaBeans
  - CORBA components
  - .NET



# Mi a szoftver komponens?

- beágyazott rendszer, autó



ECU – Electronic Control Unit

# Szoftver komponens elképzések

- Grady Booch (1987), book: *Software Components with Ada: Structures, Tools, and Subsystems*
- *“Egy újr felhasználható szoftver komponens egy logikailag összefüggő, lazán csatolt modul, ami egyetlen absztrakciót fejez ki.”*
- nem veszi figyelembe a komponens környezetet
- független telepíthetőséget nem említi
- forrás szintű komponensekre gondol

# Szoftver komponens elképzések

- Oscar Nierstrasz and Dennis Tsichritzis (1992, 1995), *Communications of the ACM* cikk
- *“Egy szoftver komponens egy csatolókkal rendelkező statikus absztrakció.”*
- csatoló: be-és kimeneti interfész
- statikus: tárházban (repository) tárolható
- független telepíthetőséget nem említi
- rendszerépítés idejében történő kompozíció
- (piaci kérdéseket nem említene)



# Szoftver komponens elképzések

- Gio Wiederhold, Peter Wegner, and Stefano Ceri (1992), cikk: Megaprogramming
- *“A megaprogramozás megamoduloknak nevezett nagy modulokkal dolgozó programozási technológia, ahol a megamodulok nagy szervezetek (mint bankok, repülőjárat foglaló rendszerek és városi közlekedési rendszerek) által nyújtott szolgáltatások funkcionalitását ragadják meg.”*
- a megamodul itt részrendszert jelent, ami nagy komponensnek felel meg

# Szoftver komponens elképzelések

- Gio Wiederhold, Peter Wegner, and Stefano Ceri (1992) (folytatás)
- *“A megamodulok belül homogén, függetlenül fenntartott szoftver rendszerek [...]. Minden megamodul leírja a kívülről elérhető adatstruktúráit és műveleteit, és belül konzisztens a viselkedése.”*
- függetlenül fejlesztett
- interfésszel rendelkezik

# Szoftver komponens elképzések

- Gio Wiederhold, Peter Wegner, and Stefano Ceri (1992) (folytatás)
- *“A megamodulokat [...] egy közösség menedzseli a saját terminológiájával, céljaival, tudásával és programozási tradíciójával. [...] Egy megamodul fogalmait, terminológiáját és interpretációs paradigmáját a megamodul ontológiájának hívják.”*
- közösség fejleszti, tehát általánosabb tudást reprezentál
- ontológia, lásd ismeretalapú modellezés

# Szoftver komponens elképzelések

- Ivar Jacobson (1993), book: *Object-Oriented Software Engineering*
- *“Komponensek alatt már megvalósított egységeket értünk, amiket a programozási nyelv konstrukcióinak javítására használunk fel. Ezeket programozás során használják fel és megfelelnek az építőipar komponenseinek.”*
- ebbe belefér a makró és a template is
- ez a csak programkészítéskor felhasználható komponens, míg manapság a telepítési idejű tulajdonságokat is beleértik

# Szoftver komponens elképzelések

- Meta Group (1994), white paper
- *“A szoftver komponensek olyan előre elkészített, letesztelt, önálló, újrafelhasználható szoftver modulok , adat és eljárás csomagok, amelyek valamilyen specifikus funkciót hajtanak végre.”*
- komponenshez nem feltétlen kell adatnak és állapotnak tartoznia
- szolgáltatásokat és környezeti függőséget nem említ
- eljárások csomagja akkor komponens, ha nem hív külső eljárásokat, csak dokumentált módon

# Szoftver komponens elképzelések

- UML 1.3 Standard (1999)
- *“Komponens: egy fizikai, helyettesíthető része egy rendszernek, ami becsomagolja a megvalósítást és interfészek egy halmazának a megvalósítását nyújtja. Egy komponens egy rendszer fizikai megvalósítását reprezentálja, beleértve a szoftver kódot (forrás, bináris vagy végrehajtható) vagy ami ezzel egyenértékű, mint szkriptek vagy parancsfájlok.”*
- a nem végrehajtható (forrás) kód is beletartozik
- nem foglalkozik az erőforrásokkal és az elvárt interfészekkel

# Szoftver komponens elképzések

- Desmond D'Souza and Alan Wills (1999)
- *“Komponens (kód) – Egy koherens szoftver megvalósítás csomag, ami (a) függetlenül fejleszthető és szállítható, (b) explicit és jól specifikált interfészei vannak a nyújtott szolgáltatásaira, (c) explicit és jól specifikált interfészei vannak a másoktól elvárt szolgáltatásokra, (d) komponálni lehet más komponensekkel, esetleg némely tulajdonságaik testre szabásával, de nem megváltoztatva magukat a komponenseket.”*
- nagyjából megfelel annak, amit itt szoftver komponensnek tekintünk

# Szoftver komponens elképzések

- CBSE Handbook (2001)
- *“Egy szoftver komponens egy olyan szoftver elem, ami egy komponens modellnek megfelel, és amit egy kompozíciós szabványnak megfelelően változtatás nélkül és függetlenül lehet kompozícióban felhasználni. A komponens modell definiálja az interakciós és kompozíciós szabályokat. A komponens modell megvalósítása a modellnek megfelelő komponensek végrehajtásának támogatásához szükséges végrehajtható szoftver halmaz.”*
- megjelenik a komponens modell fogalma



# Szyperski féle szoftver komponens definíció

ez a  
definíciónk

- Szyperski (Component Software, beyond OO programming)
- „A szoftver komponens
  - egy kompozíciós egység, aminek
  - szerződéssel definiált interfészei
  - és csak explicit környezeti függőségei vannak.
- A szoftver komponenst
  - függetlenül lehet telepíteni és
  - független fél használhatja fel kompozícióban.”

technikai  
szempont

piaci  
szempont

# Component definitions

---



## ✧ Councill and Heinmann:

- *A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.*

## ✧ 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



Component characteristic	Description
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.
Deployable	To be deployable, a component has to be self-contained. It must be able to operate as a stand-alone entity on a component platform that provides an implementation of the component model. This usually means that the component is binary and does not have to be compiled before it is deployed. If a component is implemented as a service, it does not have to be deployed by a user of a component. Rather, it is deployed by the service provider.

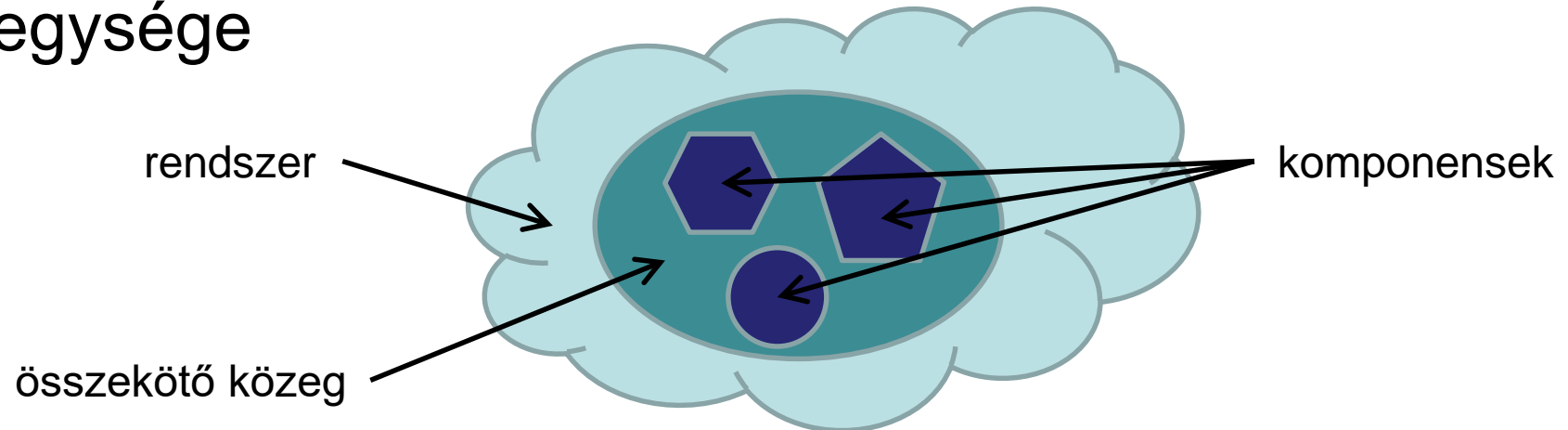
# Component characteristics



Component characteristic	Description
Documented	Components have to be fully documented so that potential users can decide whether or not the components meet their needs. The syntax and, ideally, the semantics of all component interfaces should be specified.
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.
Standardized	Component standardization means that a component used in a CBSE process has to conform to a standard component model. This model may define component interfaces, component metadata, documentation, composition, and deployment.

# Szyperski féle szoftver komponens definíció

- „Egy szoftver komponens egy **kompozíciós egység**, aminek szerződéssel definiált interfészei és csak explicit környezeti függőségei vannak. Egy szoftver komponenst függetlenül lehet telepíteni és független fél használhatja fel kompozícióban.”
- a komponens a kompozíció újrahasznosítható egysége



# Szyperski féle szoftver komponens definíció

- „Egy szoftver komponens egy kompozíciós egység, aminek **szerződéssel definiált interfészei** és csak explicit környezeti függőségei vannak. Egy szoftver komponenst függetlenül lehet telepíteni és független fél használhatja fel kompozícióban.”
- interfész: a komponens kapcsolódási felületeinek specifikációja, legfontosabb rész
- szerződés: a szolgáltatót és a klienst kölcsönösen megköti
  - funkcionális aspektusok
  - műveletek elő- és utófeltétele
  - nem funkcionális aspektusok (pl. környezeti elvárások)

# Szyperski féle szoftver komponens definíció

- „Egy szoftver komponens egy kompozíciós egység, aminek szerződéssel definiált interfészei és **csak explicit környezeti függőségei** vannak. Egy szoftver komponenst függetlenül lehet telepíteni és független fél használhatja fel kompozícióban.”
- szolgáltatót és elvárt interfész
- környezeti függőségek: a telepítési és futási környezet specifikációja
  - pl. milyen eszközökre, platformokra, erőforrásokra vagy más komponensekre van szükség

# Szyperski féle szoftver komponens definíció

- „Egy szoftver komponens egy kompozíciós egység, aminek szerződéssel definiált interfészei és csak explicit környezeti függőségei vannak. Egy szoftver komponenst **függetlenül lehet telepíteni és független fél használhatja fel** kompozícióban.”
- komponens platformra telepíthető és utána az interfészein keresztül elérhető
- a felhasználó különbözhet a fejlesztőtől és telepítőtől



# Component as a service provider



- ✧ The component is an independent, executable entity. It does not have to be compiled before it is used with other components.
- ✧ The services offered by a component are made available through an interface and all component interactions take place through that interface.
- ✧ The component interface is expressed in terms of parameterized operations and its internal state is never exposed.

# Component interfaces



## ✧ Provides interface

- Defines the services that are provided by the component to other components.
- This interface, essentially, is the component API. It defines the methods that can be called by a user of the component.

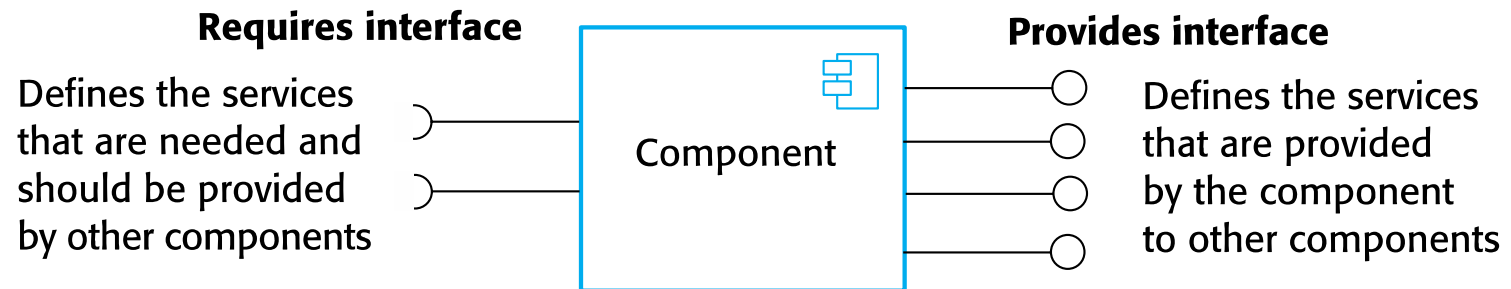
## ✧ Requires interface

- Defines the services that specifies what services must be made available for the component to execute as specified.
- This does not compromise the independence or deployability of a component because the 'requires' interface does not define how these services should be provided.

# Interfészek

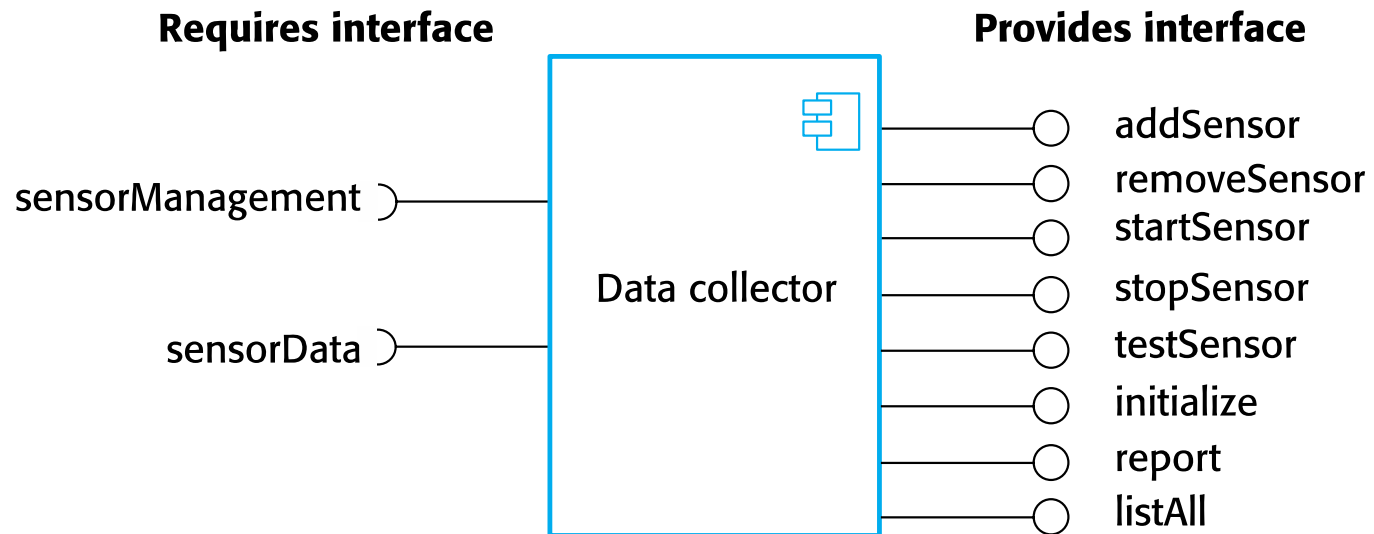
- Szolgáltatott interfész.
  - Szolgáltatások és viselkedések gyűjteménye, melyeket a komponens biztosít a kliensei számára.
  - A komponens vezérlésének belépési pontja.
- Megkövetelt, elvárt interfész.
  - Szolgáltatások és viselkedések, melyeket a komponens elvár a környezetétől.
  - Korrekt környezeti támogatás hiányában nem garantált, hogy a komponens biztosítani tudja a szolgáltatott interfészben definiált funkcionalitást a kliensei számára.

# Component interfaces



Note UML notation. Ball and sockets can fit together.

# A model of a data collector component



# A műveletek elő- és utófeltételei

- Egy művelet funkcionalitása a művelet elő- és utófeltételétől függ
- Egy előfeltétel egy invariáns, melynek igaznak kell lennie ahhoz, hogy a komponens garantálnia tudja az utófeltétel teljesülését. Előfeltételek lehetnek:
  - bemeneti paraméterekre vonatkozó korlátozások,
  - definiálhatják a komponensnek a művelet hívása előtti korrekt kezdeti állapotát.
- Az utófeltétel tartalmazza a művelet végrehajtása után a kimeneti paraméterek megengedett értéktartományát, illetve a művelet végrehajtása utáni végállapotot.
- Az elő- és utófeltételek a művelet aktuális paraméterekkel történő hívásával együtt egy állapotátmenetet definiál a művelet hívása előtti állapotból a művelet befejezése utáni végállapotba.
- Az állapot belső jellemző és kívülről csak az eredményen keresztül figyelhető meg.

# Component access



- ✧ Components are accessed using remote procedure calls (RPCs).
- ✧ Each component has a unique identifier (usually a URL) and can be referenced from any networked computer.
- ✧ Therefore it can be called in a similar way as a procedure or method running on a local computer.

# Kliens szerver kapcsolat

- A szolgáltatott interfész szempontjából egy komponens szerverként működik a környezete számára
- A megkövetelt interfész szempontjából egy komponens kliensként működik a környezete számára
- A szolgáltatott és megkövetelt interfészek között szerződések biztosítják az együttműködést.



# Szyperski féle definíció további finomításai

- minőségi kérdések
- komponens meta-modell
- komponens implementációja
- komponens modell

# Minőségi attribútumok

- Követelményeket fogalmaznak meg a komponenssel szemben
  - Teljesítmény
  - Függőségek
- A kliens csak akkor veheti igénybe a szerver szolgáltatásait, ha az a megfelelő minőségi attribútumokat biztosítani tudja.

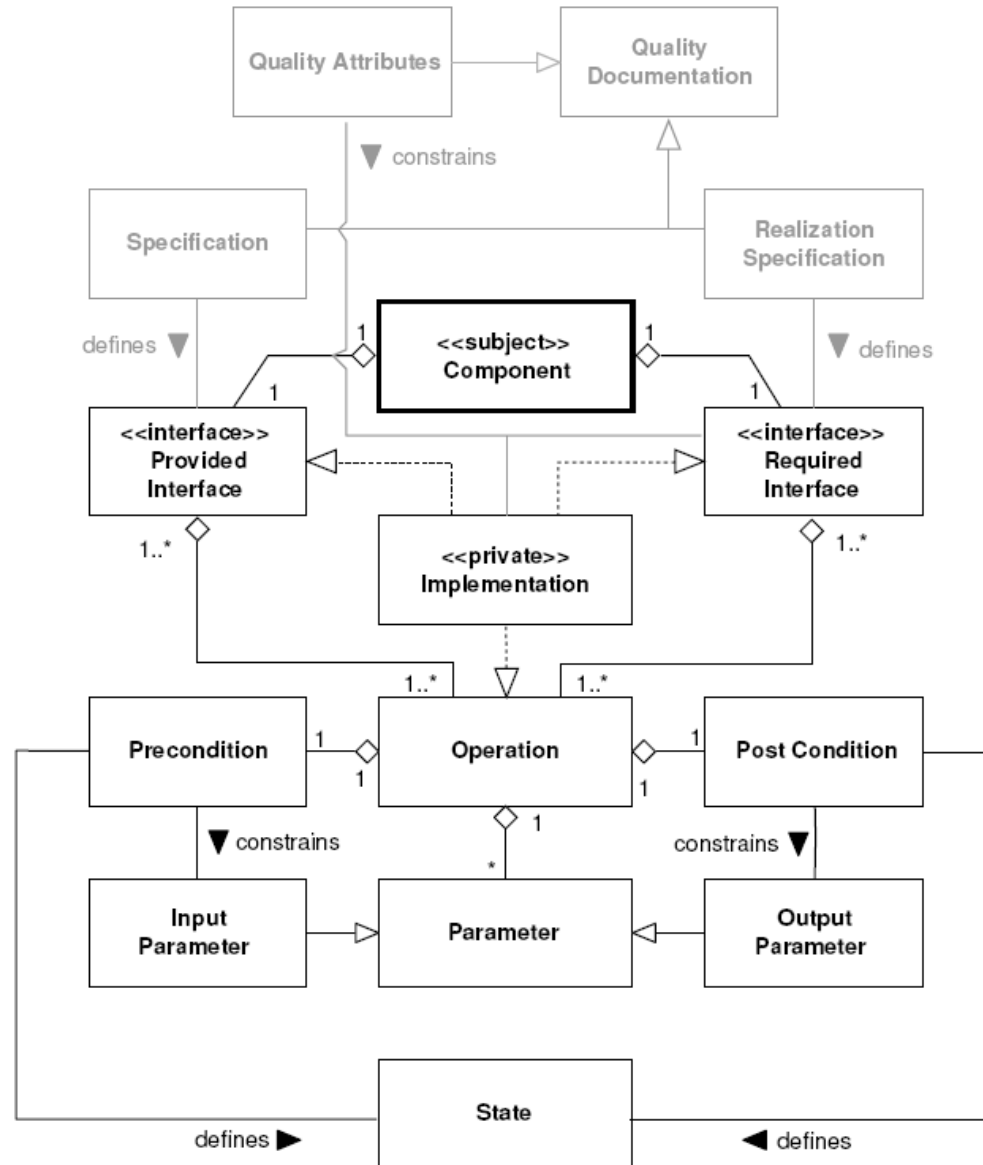
# Minőségi követelmények dokumentálása

- A komponens specifikációnak, illetve a specifikáció finomításának a részét képezi.
- Sokszor szükség van rá, mert a specifikáció túlságosan absztrakt, nehéz megérteni például, hogy egy komponens műveletét, hogyan kell hívni, vagy alkalmazni.
- Különösen hasznos dokumentálni a műveletek szekvenciáinak és kombinációinak a hatását a teljes működés szempontjából.
- Mélyebb betekintést ad a komponens használatához.

# Komponens meta-modell

- A következő ábrán egy UML meta-modell segítségével foglaljuk össze a komponens fogalmát és kapcsolatait.
- A meta-modell egy modell modellje, ami nem egy fizikai komponenst ír le, hanem csak egy koncepciót, amely alapján a fizikai komponensek megkomponálhatók.
- Az ábrán egy olyan komponenst definiálunk, aminek egy szolgáltatott és egy megkövetelt interfésze van.
- Az interfészeknek publikusnak kell lenni.

# Komponentens meta-modell



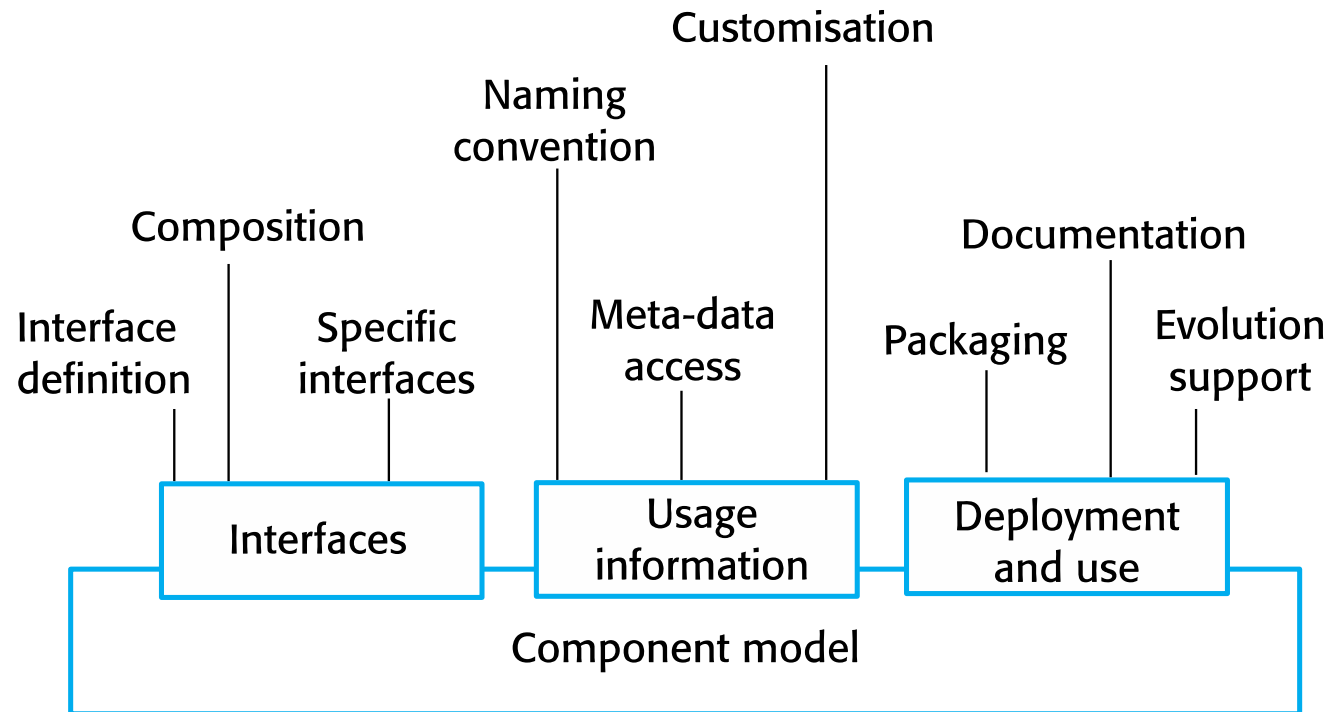
# Component models

---



- ✧ A component model is a definition of standards for component implementation, documentation and deployment.
- ✧ 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.

# Basic elements of a component model



# Elements of a component model



## ✧ Interfaces

- Components are defined by specifying their interfaces. The component model specifies how the interfaces should be defined and the elements, such as operation names, parameters and exceptions, which should be included in the interface definition.

## ✧ Usage

- In order for components to be distributed and accessed remotely, they need to have a unique name or handle associated with them. This has to be globally unique.

## ✧ Deployment

- The component model includes a specification of how components should be packaged for deployment as independent, executable entities.



# Middleware support



- ✧ Component models are the basis for middleware that provides support for executing components.
- ✧ Component model implementations provide:
  - Platform services that allow components written according to the model to communicate;
  - Support services that are application-independent services used by different components.
- ✧ To use services provided by a model, components are deployed in a **container**. This is a set of interfaces used to access the service implementations.

# Middleware services defined in a component model



## Support services

Component  
management

Concurrency

Transaction  
management

Persistence

Resource  
management

Security

## Platform services

Addressing

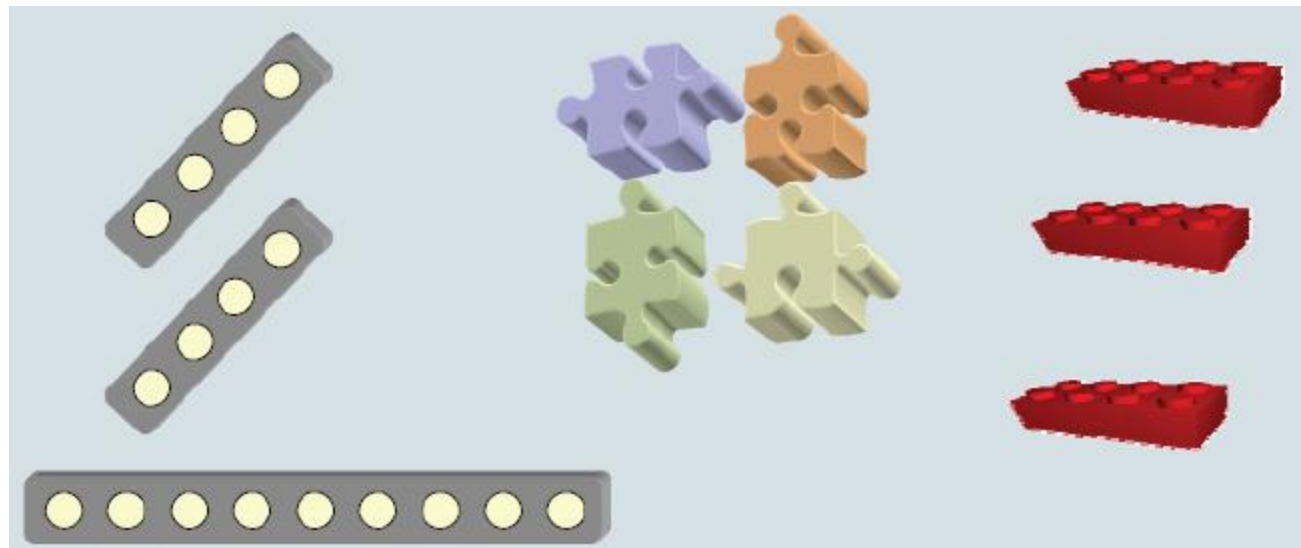
Interface  
definition

Exception  
management

Component  
communications

# Miért kell komponens modell?

- a komponenseket valamilyen komponens modellen belül lehet összekapcsolni



# Komponens modell

- definíció:  
a komponens modell azon szabványok és konvenciók specifikációja, amelyek szükségesek ahhoz, hogy az egymástól függetlenül fejlesztett komponensek kompozícióját elő lehessen állítani
- tipikusan
  - kompozíciós mechanizmusok
  - a sikeres együttműködéshez a komponensek által betartandó konvenciók

# Komponens technológia

- definíció:  
a komponens technológia a komponens modell megvalósítása a következők segítségével
  - szabványok és eljárási útmutatók a szoftver komponensek megvalósítására és működtetésére
  - szoftver eszközök a komponensek megvalósítására, összeállítására és működtetésére
- példák
  - (D)COM, .Net, Corba Component Model, Enterprise Java Bean, ...

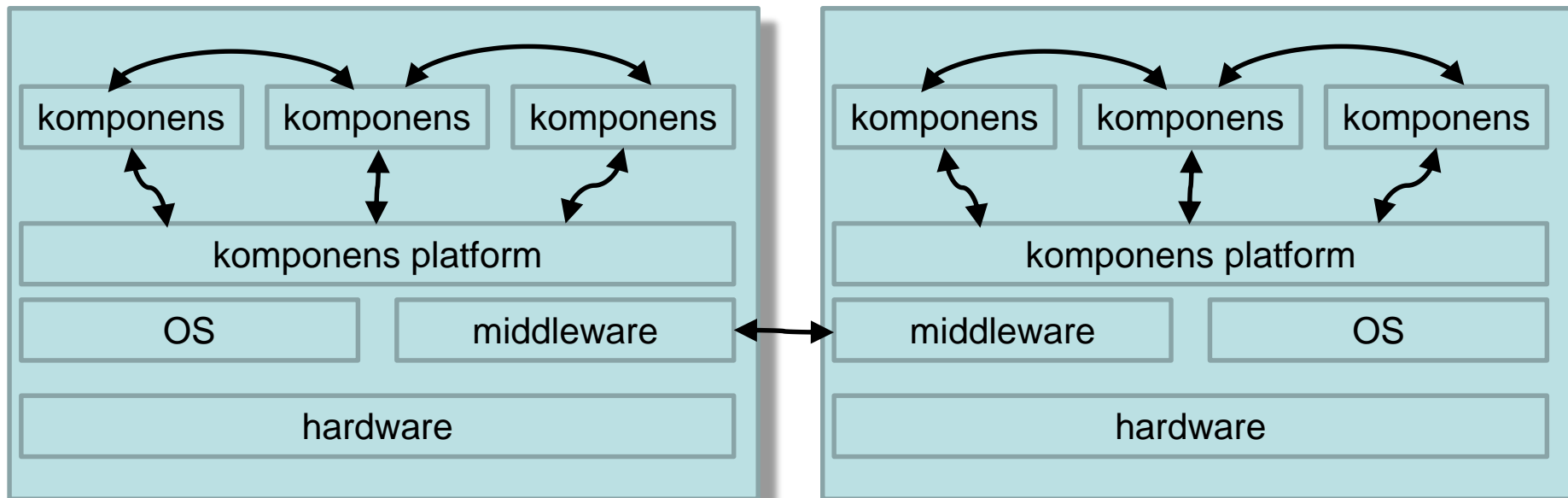
# Komponens platform

- a komponens platform a komponens modell futtatási infrastruktúrája
- támogatja
  - komponensek közötti szolgáltatásokat (kapcsolódás, interakció)
  - komponens élethciklust (létrehozás, telepítés, elindítás, leállítás, eltávolítás, kicserélés)
  - extra-funkcionális jellemzők (nyelvi/OS interoperabilitás, ütemezés, szolgáltatás minőség menedzsment, terhelés elosztás, biztonság, hibatűrés/replikáció, ...)

# Komponens keretrendszer

- a komponens modell több, esetleg opcionális keretrendszeret nyújt
- a keretrendszer különböző szolgáltatások igénybevételéhez a platformhoz való kapcsolódást biztosítja, mint pl.
  - platform elérése (dokkolás)
  - másik komponenshez kapcsolódás
  - hiba kezelés
  - tranzakciók
  - ...

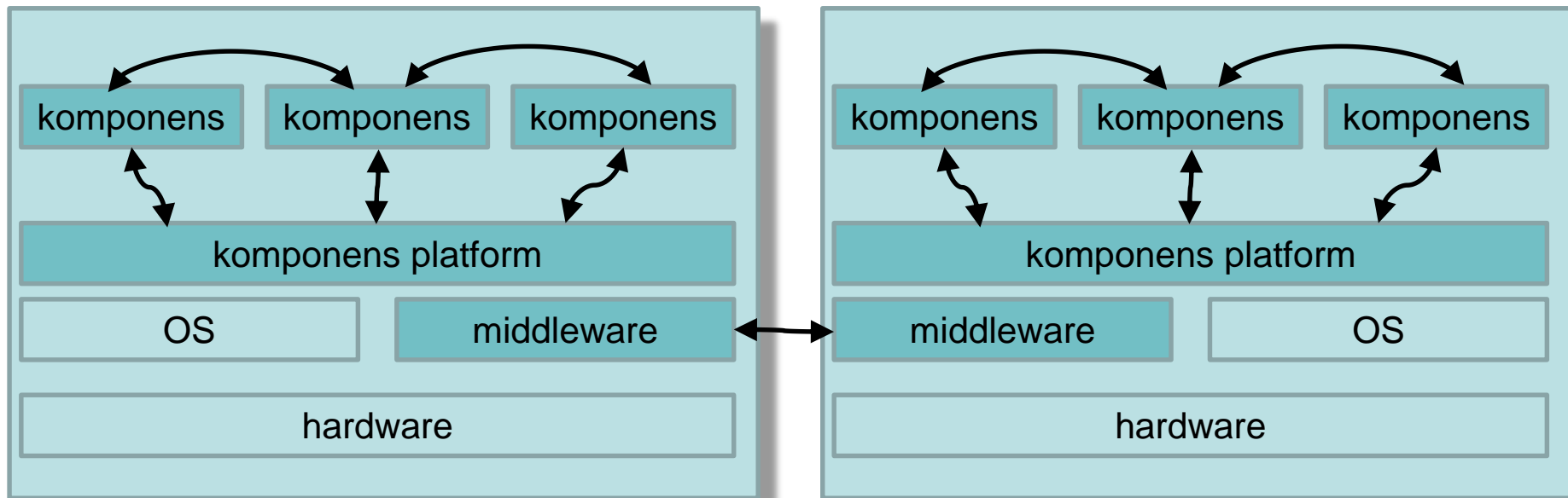
# Komponens modell architektúra





# Komponens modell architektúra

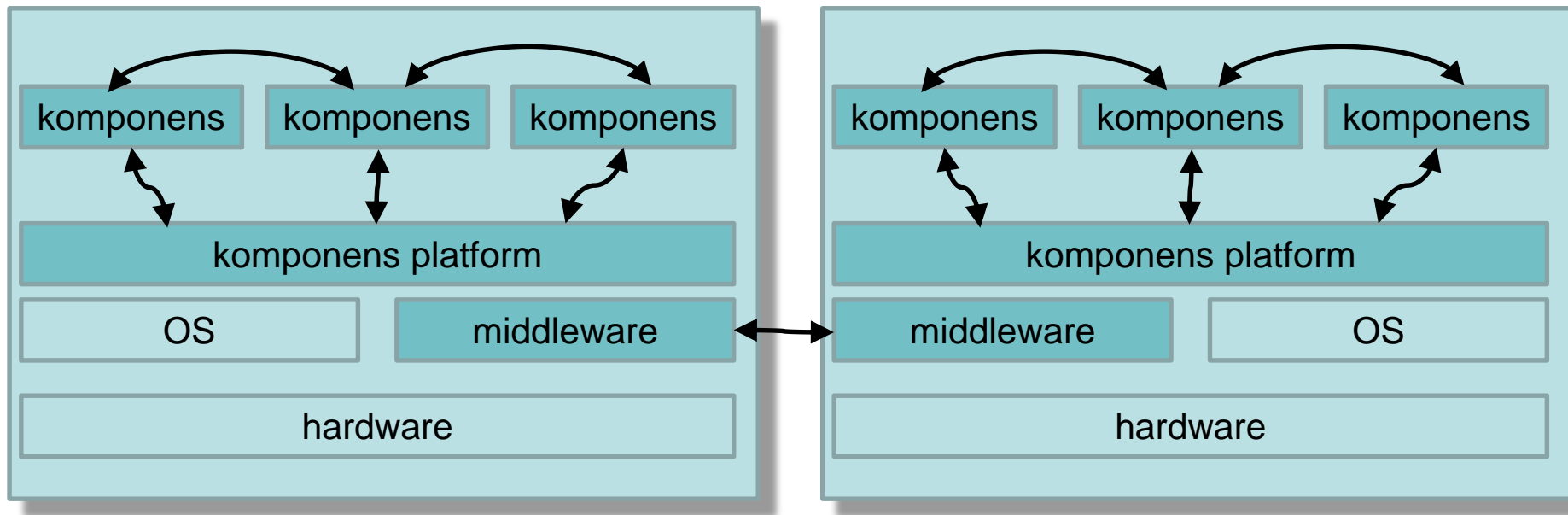
- Corba



- eredeti célja OO alkalmazások együttműködése (middleware), a komponens modell ebből származik

# Komponens modell architektúra

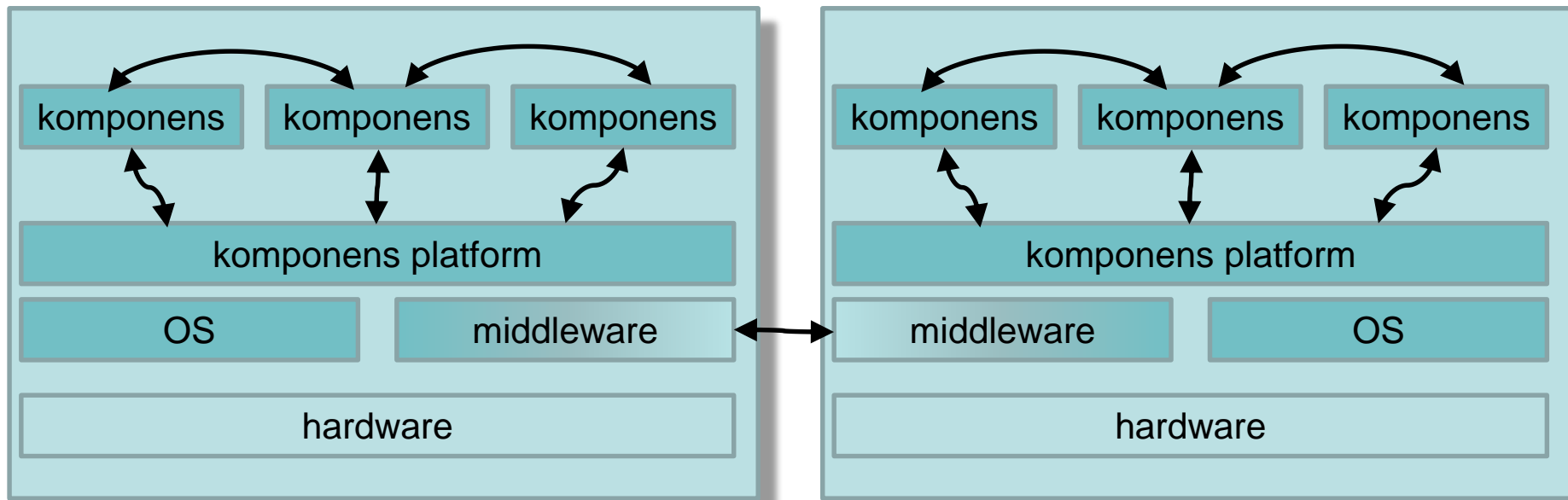
- EJB



- további szolgáltatások, pl. tranzakció

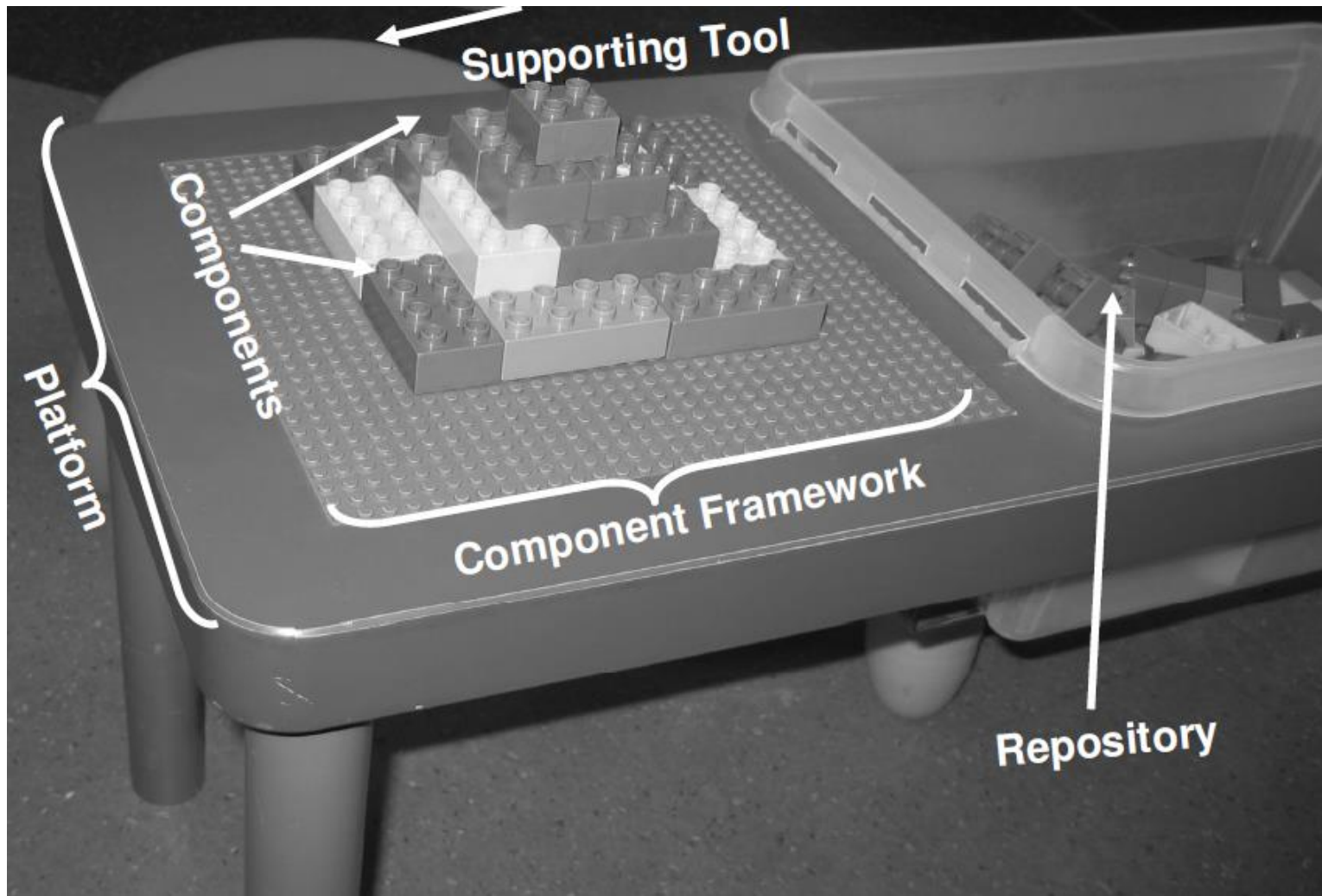
# Komponens modell architektúra

- DCOM, .Net



- komponens támogatás az OS-ben
- hálózati támogatás fokozatosan

# Komponens modell analógia



# CBSE processes

# CBSE processes

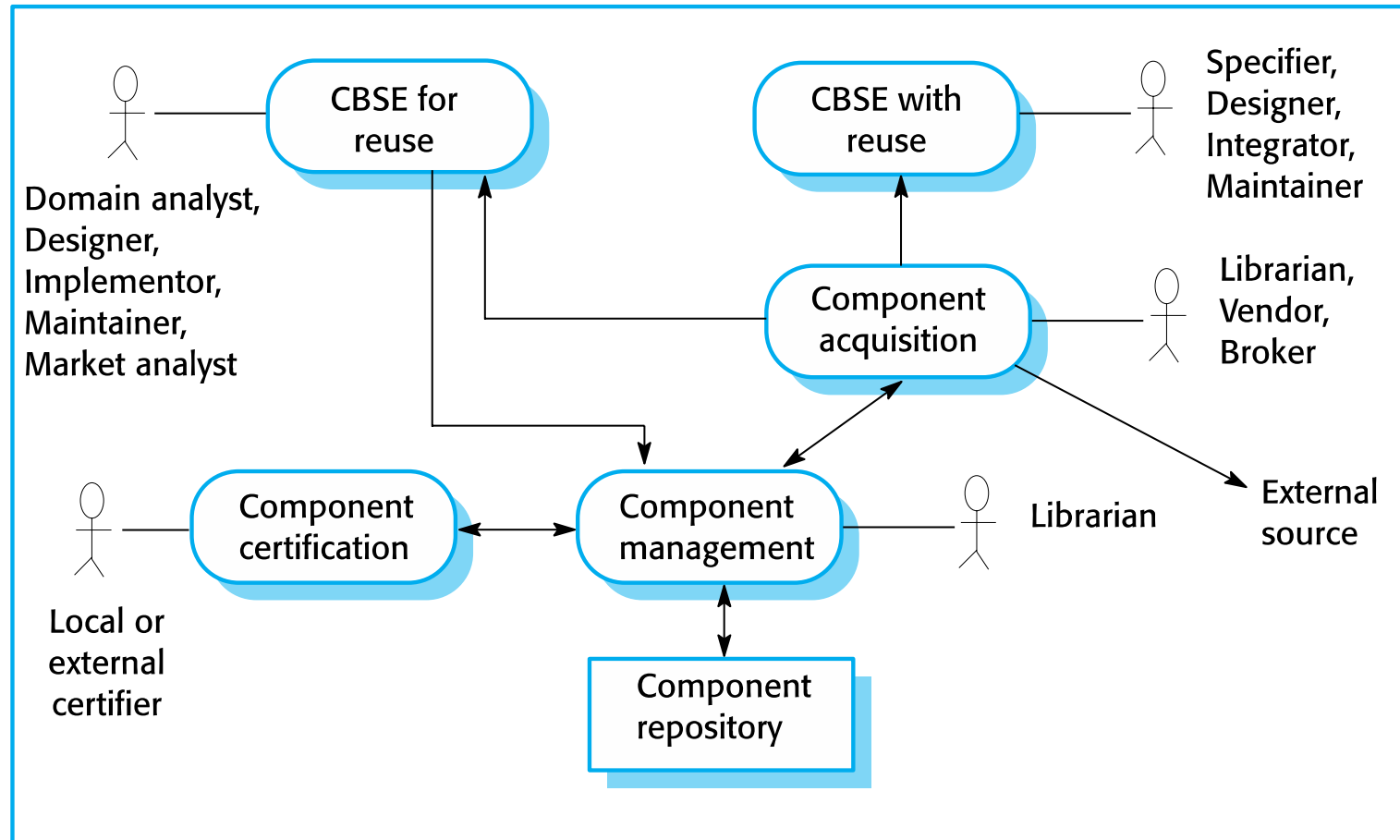


- ✧ CBSE processes are software processes that support component-based software engineering.
  - They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.
- ✧ Development for reuse
  - This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
- ✧ Development with reuse
  - This process is the process of developing new applications using existing components and services.

# CBSE processes



## CBSE processes



# Supporting processes



- ✧ Component acquisition is the process of acquiring components for reuse or development into a reusable component.
  - It may involve accessing locally- developed components or services or finding these components from an external source.
- ✧ Component management is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored and made available for reuse.
- ✧ Component certification is the process of checking a component and certifying that it meets its specification.



# CBSE for reuse

---



- ✧ CBSE for reuse focuses on component development.
- ✧ Components developed for a specific application usually have to be generalised to make them reusable.
- ✧ A component is most likely to be reusable if it associated with a stable domain abstraction (business object).
- ✧ For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

# Component development for reuse



- ✧ Components for reuse may be specially constructed by generalising existing components.
- ✧ Component reusability
  - Should reflect stable domain abstractions;
  - Should hide state representation;
  - Should be as independent as possible;
  - Should publish exceptions through the component interface.
- ✧ There is a trade-off between reusability and usability
  - The more general the interface, the greater the reusability but it is then more complex and hence less usable.

# Changes for reusability

---



- ✧ Remove application-specific methods.
- ✧ Change names to make them general.
- ✧ Add methods to broaden coverage.
- ✧ Make exception handling consistent.
- ✧ Add a configuration interface for component adaptation.
- ✧ Integrate required components to reduce dependencies.

# Exception handling



- ✧ Components should not handle exceptions themselves, because each application will have its own requirements for exception handling.
  - Rather, the component should define what exceptions can arise and should publish these as part of the interface.
- ✧ In practice, however, there are two problems with this:
  - Publishing all exceptions leads to bloated interfaces that are harder to understand. This may put off potential users of the component.
  - The operation of the component may depend on local exception handling, and changing this may have serious implications for the functionality of the component.

# Reusable components



- ✧ The development cost of reusable components may be higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost.
- ✧ Generic components may be less space-efficient and may have longer execution times than their specific equivalents.

# Legacy system components



- ✧ Existing legacy systems that fulfil a useful business function can be re-packaged as components for reuse.
- ✧ This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.
- ✧ Although costly, this can be much less expensive than rewriting the legacy system.

# Component management



- ✧ Component management involves deciding how to classify the component so that it can be discovered, making the component available either in a repository or as a service, maintaining information about the use of the component and keeping track of different component versions.
- ✧ A company with a reuse program may carry out some form of component certification before the component is made available for reuse.
  - Certification means that someone apart from the developer checks the quality of the component.

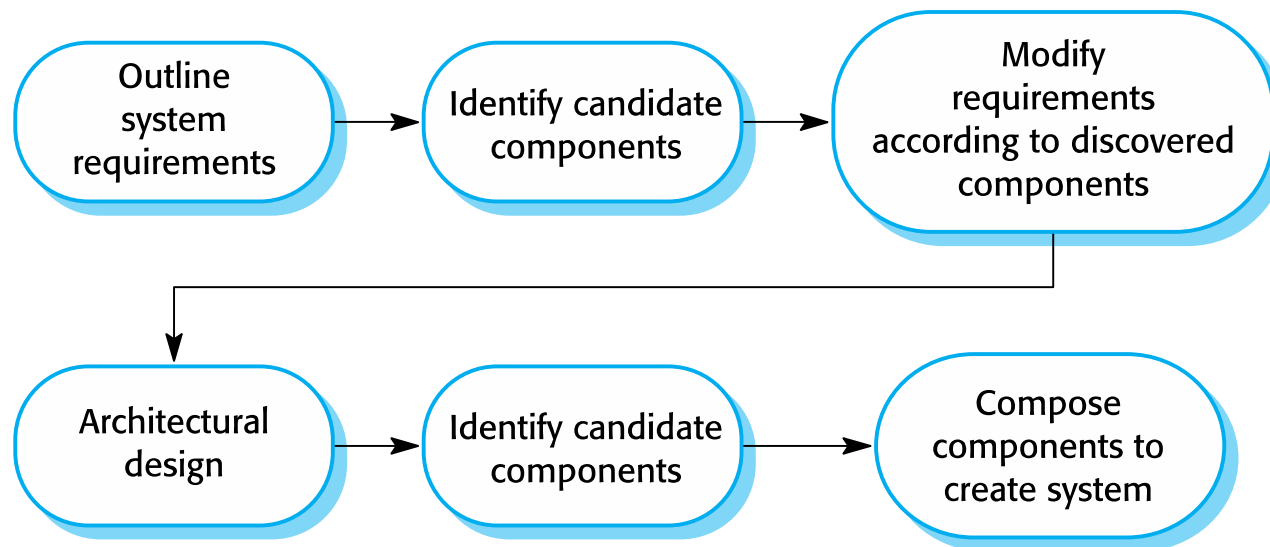
# CBSE with reuse



- ✧ CBSE with reuse process has to find and integrate reusable components.
- ✧ When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.
- ✧ This involves:
  - Developing outline requirements;
  - Searching for components then modifying requirements according to available functionality.
  - Searching again to find if there are better components that meet the revised requirements.
  - Composing components to create the system.



# CBSE with reuse



# The component identification process

---



# Component identification issues



- ✧ **Trust.** You need to be able to trust the supplier of a component. At best, an untrusted component may not operate as advertised; at worst, it can breach your security.
- ✧ **Requirements.** Different groups of components will satisfy different requirements.
- ✧ **Validation.**
  - The component specification may not be detailed enough to allow comprehensive tests to be developed.
  - Components may have unwanted functionality. How can you test this will not interfere with your application?

# Component validation



- ✧ Component validation involves developing a set of test cases for a component (or, possibly, extending test cases supplied with that component) and developing a test harness to run component tests.
  - The major problem with component validation is that the component specification may not be sufficiently detailed to allow you to develop a complete set of component tests.
- ✧ As well as testing that a component for reuse does what you require, you may also have to check that the component does not include any malicious code or functionality that you don't need.

# Ariane launcher failure – validation failure?



- ✧ In 1996, the 1st test flight of the Ariane 5 rocket ended in disaster when the launcher went out of control 37 seconds after take off.
- ✧ The problem was due to a reused component from a previous version of the launcher (the Inertial Navigation System) that failed because assumptions made when that component was developed did not hold for Ariane 5.
- ✧ The functionality that failed in this component was not required in Ariane 5.



# Component composition

# Component composition

---



- ✧ The process of assembling components to create a system.
- ✧ Composition involves integrating components with each other and with the component infrastructure.
- ✧ Normally you have to write 'glue code' to integrate components.

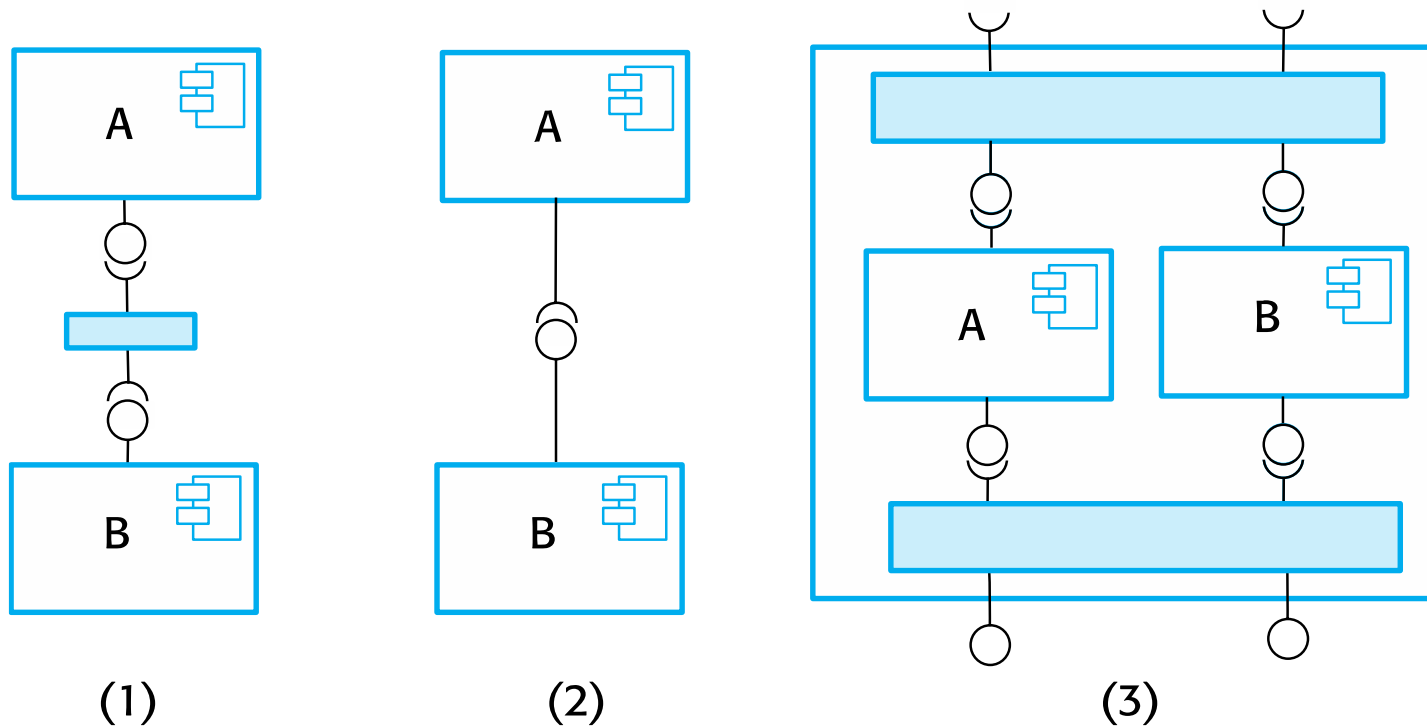
# Types of composition



- ✧ **Sequential composition** (1) where the composed components are executed in sequence. This involves composing the provides interfaces of each component.
- ✧ **Hierarchical composition** (2) where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.
- ✧ **Additive composition** (3) where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.



# Types of component composition



# Glue code



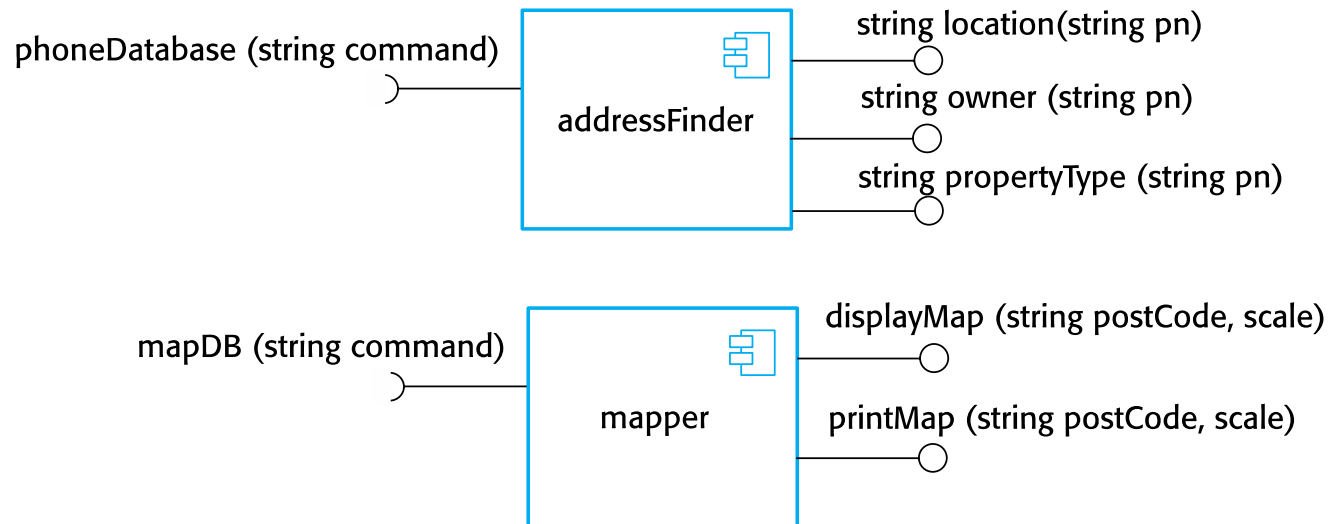
- ✧ Code that allows components to work together
- ✧ If A and B are composed sequentially, then glue code has to call A, collect its results then call B using these results, transforming them into the format required by B.
- ✧ Glue code may be used to resolve interface incompatibilities.

# Interface incompatibility



- ✧ **Parameter incompatibility** where operations have the same name but are of different types.
- ✧ **Operation incompatibility** where the names of operations in the composed interfaces are different.
- ✧ **Operation incompleteness** where the provides interface of one component is a subset of the requires interface of another.

# Components with incompatible interfaces



# Adaptor components



- ✧ Address the problem of component incompatibility by reconciling the interfaces of the components that are composed.
- ✧ Different types of adaptor are required depending on the type of composition.
- ✧ An addressFinder and a mapper component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component.

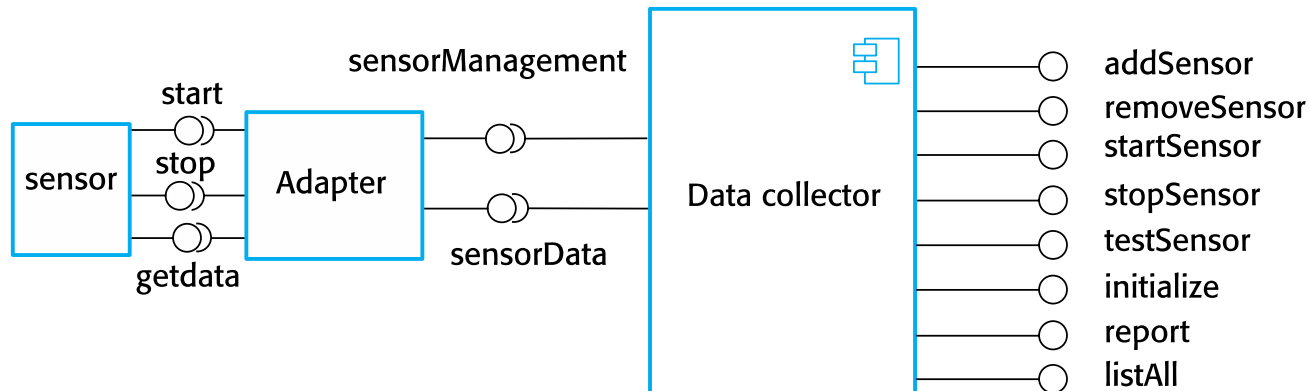
# Composition through an adaptor



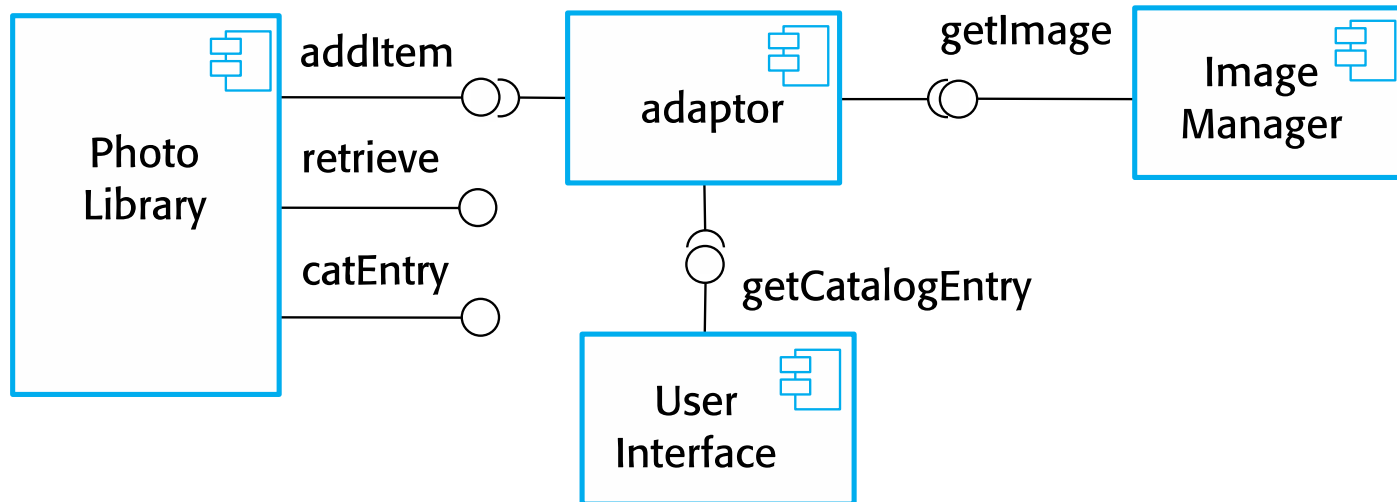
- ✧ The component `postCodeStripper` is the adaptor that facilitates the sequential composition of `addressFinder` and `mapper` components.

```
address = addressFinder.location (phonenumber) ;  
postCode = postCodeStripper.getPostCode (address) ;  
mapper.displayMap(postCode, 10000)
```

# An adaptor linking a data collector and a sensor



# Photo library composition





# Interface semantics



- ✧ You have to rely on component documentation to decide if interfaces that are syntactically compatible are actually compatible.
- ✧ Consider an interface for a PhotoLibrary component:

```
public void addItem (Identifier pid ; Photograph p; CatalogEntry photodesc) ;  
public Photograph retrieve (Identifier pid) ;  
public CatalogEntry catEntry (Identifier pid) ;
```

# Photo Library documentation



**Assume that the documentation for the addItem method in Photo Library is:**

*“This method adds a photograph to the library and associates the photograph identifier and catalogue descriptor with the photograph.”*

**This description appears to explain what the component does, but consider the following questions:**

*“what happens if the photograph identifier is already associated with a photograph in the library?”*

*“is the photograph descriptor associated with the catalogue entry as well as the photograph i.e. if I delete the photograph, do I also delete the catalogue information?”*

# The Object Constraint Language



- ✧ The Object Constraint Language (OCL) has been designed to define constraints that are associated with UML models.
- ✧ It is based around the notion of pre and post condition specification – common to many formal methods.

# The OCL description of the Photo Library interface



-- The context keyword names the component to which the conditions apply

**context** addItem

-- The preconditions specify what must be true before execution of addItem

**pre:** PhotoLibrary.libSize() > 0

PhotoLibrary.retrieve(pid) = null

-- The postconditions specify what is true after execution

**post:** libSize () = libSize()@pre + 1

PhotoLibrary.retrieve(pid) = p

PhotoLibrary.catEntry(pid) = photodesc

**context** delete

**pre:** PhotoLibrary.retrieve(pid) <> null ;

**post:** PhotoLibrary.retrieve(pid) = null

PhotoLibrary.catEntry(pid) = PhotoLibrary.catEntry(pid)@pre

PhotoLibrary.libSize() = libSize()@pre—1

# Photo library conditions



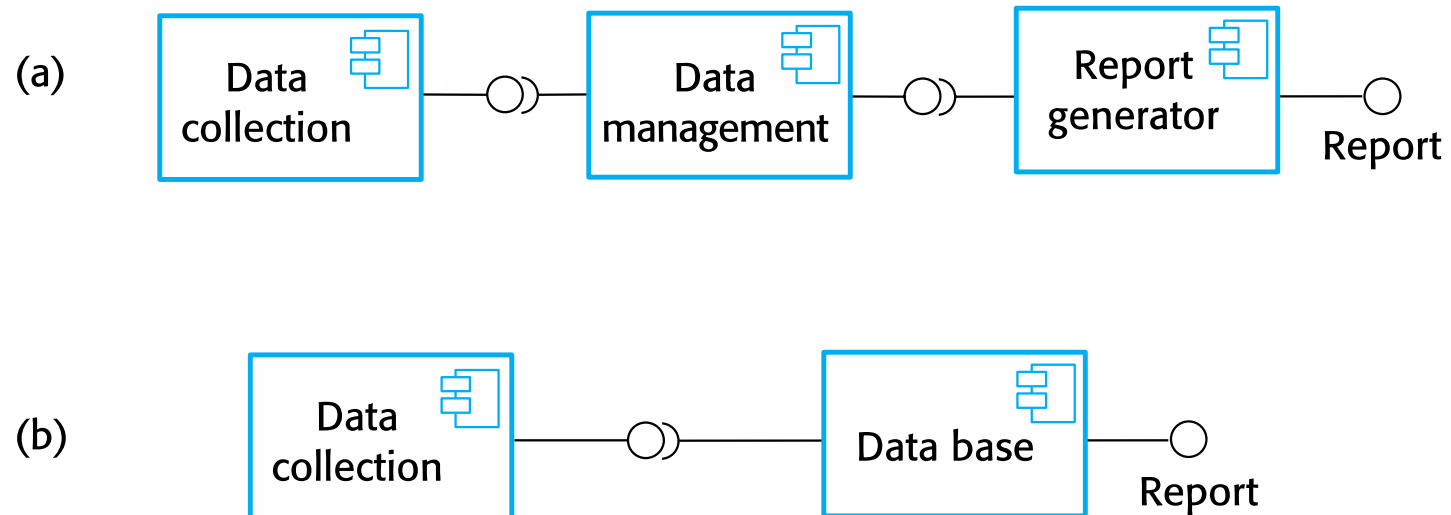
- ✧ As specified, the OCL associated with the Photo Library component states that:
- There must not be a photograph in the library with the same identifier as the photograph to be entered;
  - The library must exist - assume that creating a library adds a single item to it;
  - Each new entry increases the size of the library by 1;
  - If you retrieve using the same identifier then you get back the photo that you added;
  - If you look up the catalogue using that identifier, then you get back the catalogue entry that you made.

# Composition trade-offs



- ✧ When composing components, you may find conflicts between functional and non-functional requirements, and conflicts between the need for rapid delivery and system evolution.
- ✧ You need to make decisions such as:
  - What composition of components is effective for delivering the functional requirements?
  - What composition of components allows for future change?
  - What will be the emergent properties of the composed system?

# Data collection and report generation components



# Key points



- ✧ CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.
- ✧ A component is a software unit whose functionality and dependencies are completely defined by its interfaces.
- ✧ Components may be implemented as executable elements included in a system or as external services.
- ✧ A component model defines a set of standards that component providers and composers should follow.
- ✧ The key CBSE processes are CBSE for reuse and CBSE with reuse.



# Key points

---



- ✧ During the CBSE process, the processes of requirements engineering and system design are interleaved.
- ✧ Component composition is the process of ‘wiring’ components together to create a system.
- ✧ When composing reusable components, you normally have to write adaptors to reconcile different component interfaces.
- ✧ When choosing compositions, you have to consider required functionality, non-functional requirements and system evolution.