KobrA előadás

A KobrA komponensalapú programfejlesztési modell

I. rész

Kozma László tanár úr előadásai alapján

- eddig volt:
  - mi a komponens
  - mi a komponens modell
  - mik a komponens modell részei
  - ismert, hogyan írhatunk le komponensalapú rendszereket UML segítségével
- most jön:
  - módszer komponensalapú rendszerek tervezéséhez

Felülről – lefelé történő fejlesztés

- Az egész elkészíteni kívánt rendszert egyre kisebb alrendszerekre bontjuk
- Az önállóan kezelhető alrendszereket egyenként
  - specifikáljuk
  - megtervezzük
  - megvalósítjuk
- Az így elkészült alrendszereket rendszerbe integráljuk
- Az elkészült alrendszerek illeszkednek a projektünkhöz, de általában csak ahhoz

Komponensalapú fejlesztés

- Meglévő szoftverkomponensekből építsük fel a rendszerünket, használjuk azokat mint építőköveket.
- Ezek az építőkövek
  - kellően általánosak
  - általában korábban hoztuk létre őket
    - pontosan erre a célra vagy
    - egy másik szoftverrendszer részeként
- Tipikus bottom-up megközelítés

- Általában a kétféle megközelítés keveredik
  - Hagyományos szoftverfejlesztésnél használunk függvénykönyvtárakat, rendszerhívásokat, ezek gyakran komponensként viselkedhetnek
  - Egyszerűen összeillesztve néhány, már meglévő komponenst, valószínűleg nem pontosan az elvárt eredményt kapjuk, ezért a komponensalapú fejlesztésnél is szükségünk lesz a top-down technikák egy részére

- Napjaink általános fejlesztői gyakorlata
  - A létrehozandó rendszert dekomponáljuk kisebb részekre
  - Ezeket a részeket már létező funkcionalitású komponensekre próbáljuk leképezni, ha ilyenek már rendelkezésre állnak
  - Ha nem, akkor tovább dekomponáljuk a rendszert
  - Végül az egyes alrendszerekhez rendelt komponenseket felhasználjuk az új rendszer felépítésére vagy ha ez nem lehetséges, akkor elkészítjük a szükséges komponenseket

- Minden komoly szoftverfejlesztési projektnek valamilyen modellen kell alapulnia
- Egy jó modell eljárást definiál a következő tevékenységek végrehajtására:
  - Analízis
  - Tervezés
  - Megvalósítás (Programozás)
  - Verifikáció
  - Dokumentáció
  - Stb.

A szoftverfejlesztési modell fogalma

- Alapvetően eljárások, ajánlások gyűjteménye
  - Segíti a szoftverfejlesztés teljes ciklusát
  - Megmondja, hogy egy adott fejlesztési lépésben mit kell tenni és azt hogyan kell megtenni
  - Egyetlen mondatba sűrítve a szoftverfejlesztési modell egy recept a programok létrehozására
- Szoftverfejlesztési modellek
  - vízesés modell / V-modell
  - spirál modell
  - Komponensalapú
  - stb.
- A későbbiekben részletesebben tárgyaljuk a KobrA szoftverfejlesztési modellt

Komponens alapú fejlesztés

- A komponens alapú fejlesztés alapvetően két diszjunkt területre bomlik:
  - a komponens fejlesztés
    - a komponensek, mint egyedi építőkövek kifejlesztésére koncentrál
  - az alkalmazás fejlesztés
    - a kész komponensekből a rendszer felépítésére koncentrál.
- Egy másik megközelítés szerint
  - az alkalmazásfejlesztés a létrehozandó rendszer dekomponálásával foglalkozik
  - a komponensfejlesztés pedig azzal, hogy az egyes komponensek, hogyan illeszkednek az alkalmazásfejlesztés során létrehozott dekompozíciós hierarchiához

Komponens fejlesztés

- Az alkalmazás építőkövét, a komponenst hozza létre
- A fejlesztést a komponens szolgáltató végzi
  - Fontos szempont az újrafelhasználhatóság
  - Felelős azért, hogy mennyire lesz újrahasznosítható a komponens
  - Feladata, hogy meghatározza, hogy az egyes komponensek miként illeszkednek a logikai de-kompozíciós hierarchiába, amelyet az alkalmazás fejlesztő hoz létre

Alkalmazás fejlesztés

- Arra koncentrál, hogy a komponensek miként építhetők össze egy alkalmazássá.
- A fejlesztést a komponens felhasználója végzi.
  - A komponensek újrafelhasználására koncentrál
  - Feladata a rendszert egyre finomabb részekre felbontani
  - A dekomponálás iteratív eljárás.

A KobrA programfejlesztési modell

Hans-Gerhard Gross
Component-Based Software Testing with UML

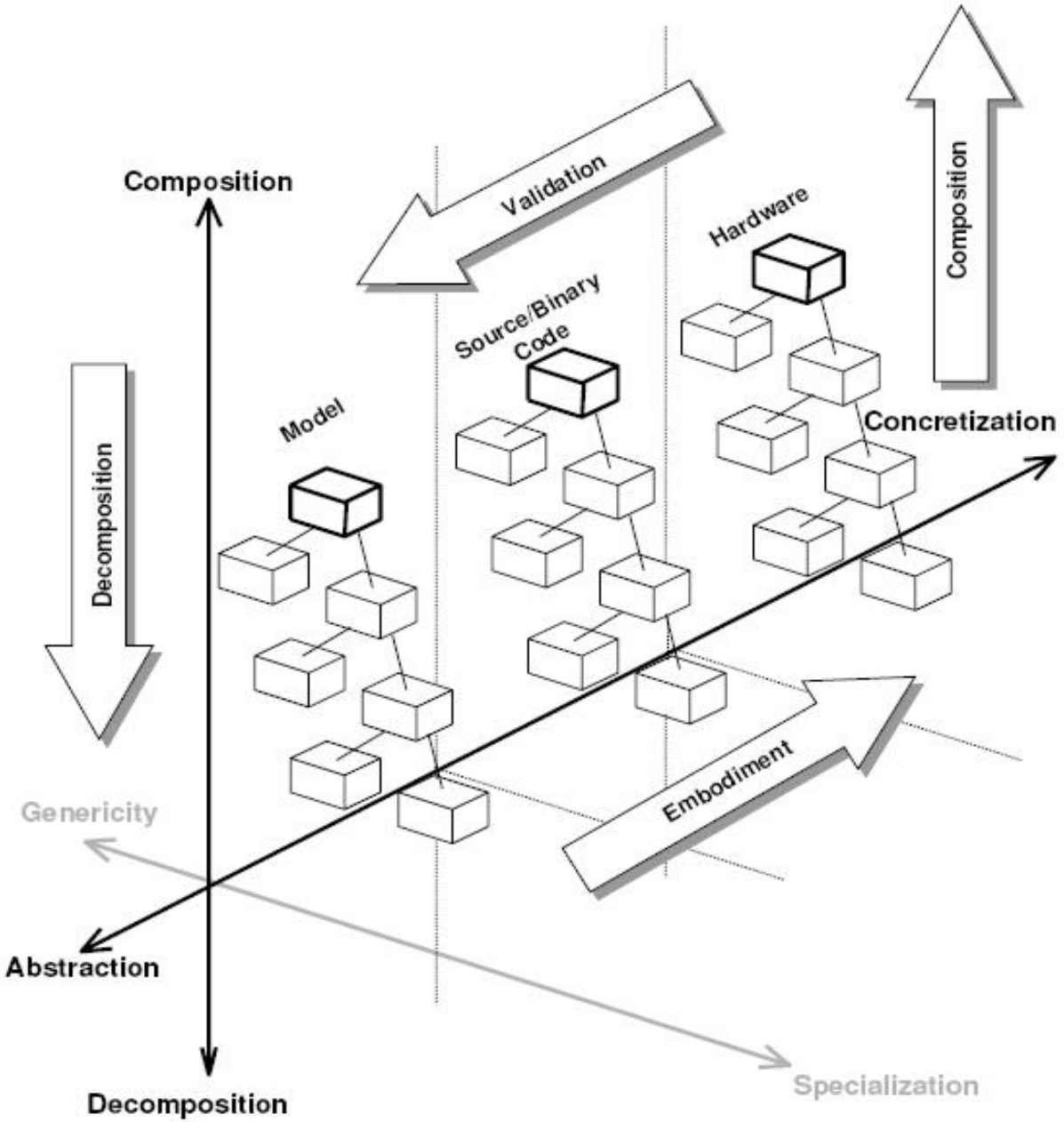
című könyve alapján

- Kifejlesztő intézet: Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Németország
- Több komponensalapú programfejlesztési modell szintézisének tekinthető:
  - » OMT
  - » Fusion
  - » ROOM
  - » HOOD
  - » OORAM
  - » Stb.

- Cél: ezen módszerek erősségeinek egyesítése, a gyengeségek kiküszöbölésével.
- A KobrA segítséget nyújt az összes tevékenységhez (analízis, tervezés, programozás, verifikáció, dokumentáció)
- Az analízis és tervezés eszköze az UML
  - az UML jó döntés, mert használatával anélkül tudunk modellezni, hogy bármilyen implementációs, vagy rendszerspecifikus megfontolást tennénk

- A KobrA-ban legfontosabb célunk, hogy a rendszer alapvető struktúráját absztrakt formában állítsuk elő, hogy sokféle implementációs technológiára leképezhető legyen.
- Legfontosabb elvek:
  - Problémák elkülönítése
  - Modellalapú fejlesztés
  - Komponensek használata
  - Objektumorientált technológiák alkalmazása
- A következő ábra bemutatja azt a három dimenziós modellt, amelyet a KobrA megalkotói javasolnak.

2.1. ábra


Composition
Validation
Composition
Hardware
Source/Binary Code
Model
Concretization
Decomposition
Embodiment
Genericity
Abstraction
Specialization
Decomposition

- A KobrA a fejlesztési folyamatot két alapvető dimenzióra osztja
- Minden tevékenységet ebben a két dimenzióban értékelünk, ebből a szemszögből vizsgálunk
- Ez négyféle lapvető irányt határoz meg a fejlesztés számára:
  - Kompozíció/Dekompozíció
  - Absztrakció/Konkretizáció

- Kompozíció/dekompozíció:
  - Dekompozíció: oszd meg és uralkodj elv alkalmazása, a rendszer alkalmas szétvágása önmagukban kezelhető részekre
  - Kompozíció: Az implementált részeket összeillesztjük
- Absztrakció/konkretizáció:
  - A felhasználók számára jól érthető modelltől közelítünk a számítógép által értelmezhető futtatható program felé

- Még egy dimenzió:
- általánosítás/specializáció
  - Ebben a dimenzióban végzett munka akkor kap szerepet, ha egy egész termékcsaládot fejlesztünk.
  - Bizonyos komponensek pontos működése határozza meg, hogy pontosan melyik termékről van szó (változatok)
  - Ezt a dimenziót most nem vizsgáljuk, mert elbonyolítja a modellünket

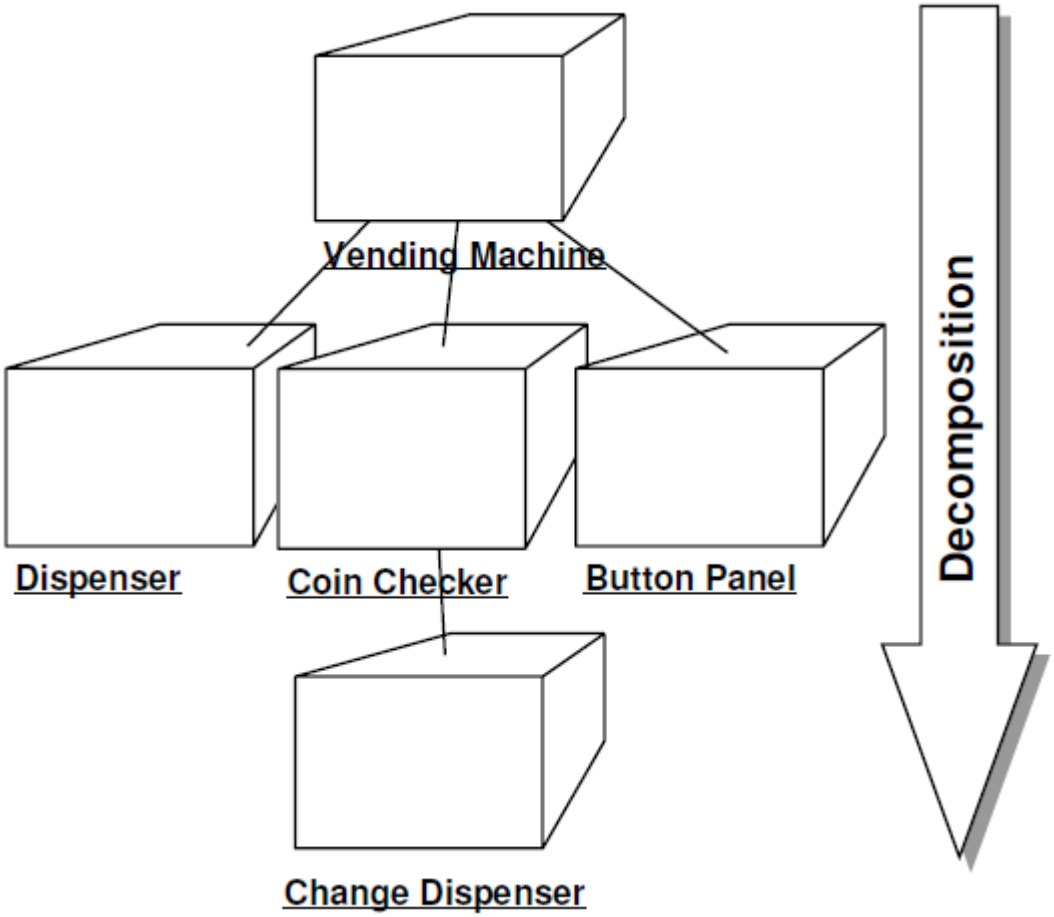
Dekompozíció

Példa a dekompozicióra

- pénzbedobós automata
  - Felépítés
    - Pénzbedobó
    - Érem ellenőrző
    - Billentyűzet a választáshoz
    - Árukiadó

- Ebben az esetben a kompozíció/dekompozíció dimenzió mentén haladunk lefelé
- A dekompozíció során olyan egységeket próbálunk azonosítani amelyek már készen vannak és újra fel tudjuk őket használni
- (vagy ha ilyenre nincs esély, akkor a megvalósítás után újra felhasználhatónak tervezzük)
- A 2.2 ábrán bemutatjuk a dekompozició eredményét az árusító automatára vonatkozóan

- 2.2 ábra


Vending Machine
Dispenser
Coin Checker
Button Panel
Change Dispenser
Decomposition

- A dekompozíció sikere a probléma jó megértésén múlik
- A célunk legyen mindig az, hogy a dekompozíció során mindig újra felhasználható komponensekhez jussunk
- Ha nem ez vezérel bennünket, akkor oda jutunk, hogy mindent meg kell írni újra, vagyis a módszerünk nem fog túlmutatni a top-down módszereken.

Megtestesítés

- Dekompozíció során azonosítjuk és specifikáljuk az egyes alrendszereket.
- A specifikációt és a megvalósítást elkülönítjük.
- Specifikációban a felhasználó által elvárt és a számára nyújtott szolgáltatásokat nagyon absztrakt szinten írjuk le.
- Megvalósítás szintjén a felhasználói interfészben specifikált szolgáltatások tényleges megvalósítását adjuk meg.
- A számítógépen futtatható megoldást az UML modellből kiindulva konkrét reprezentációkon keresztül érhetjük el. Ezt a folyamatot hívjuk megtestesítésnek (embodiment).

- A pénzbedobós automatánk esetében a **megtestesítés** folyamata nemcsak kódolást jelent, hanem mivel ez egy beágyazott rendszernek tekinthető, annak eldöntését is, hogy mely részeket valósítsuk meg hardverrel és melyeket szoftverrel.

Kompozíció

- Egyes részeket megvalósítottunk
- Másokat újrafelhasználunk
- Ezután az egyes részeket egésszé rakjuk össze, az alrendszereket rendszerré integráljuk.
- Ez egy felfele mozgás a kompozíció/dekompozíció dimenzióban

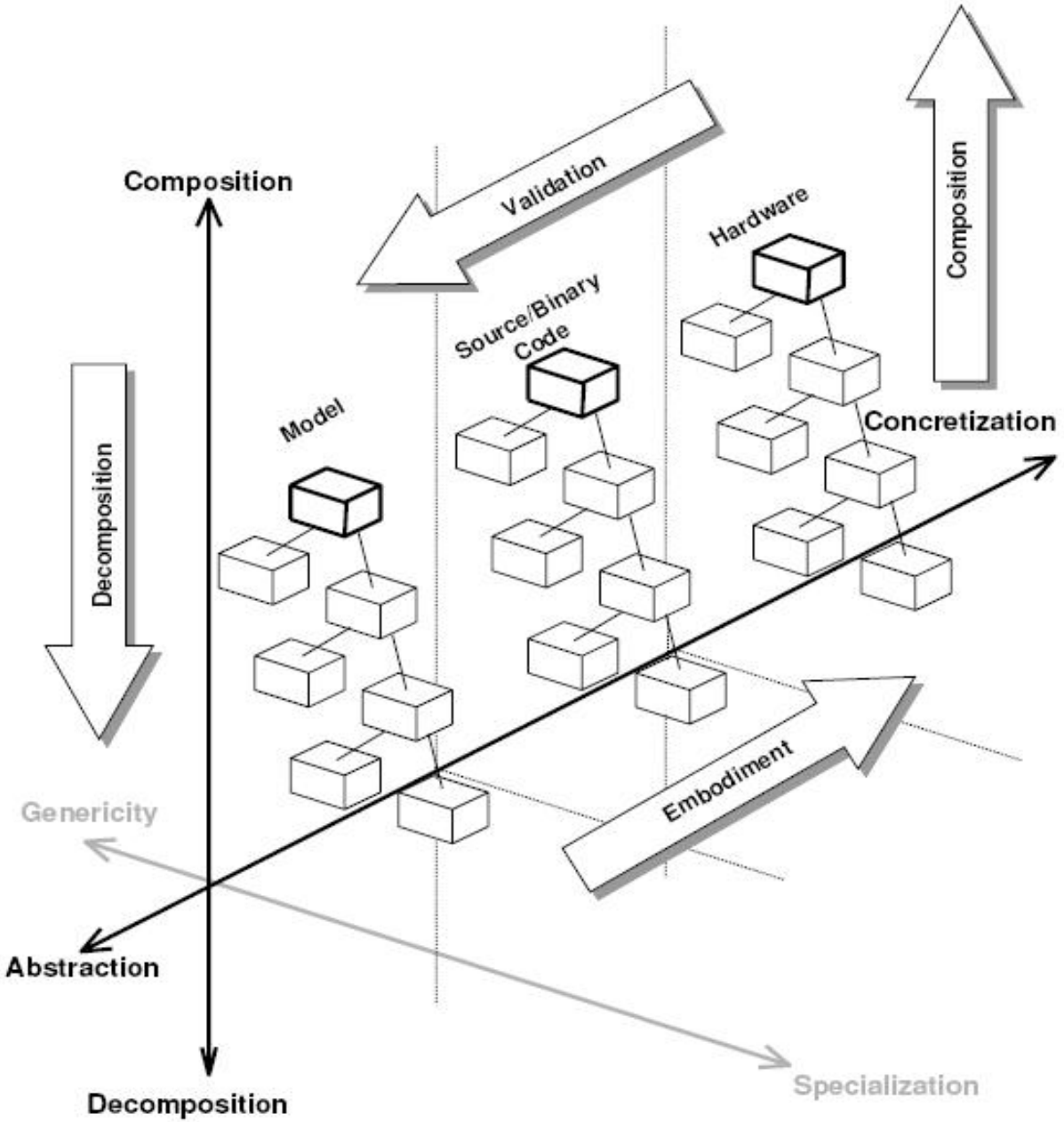
Érvényesítés

- A érvényesítés (validation) az implementált elemek összevetését jelenti az absztrakt modellel. Ez az absztrakció/konkretizáció dimenzió irányában való mozgást jelent
- Nem feltétlenül az utolsó művelet:
  - Nem kell teljesen dekomponálni a rendszert, hogy részeket implementálhassunk
  - Egy implementált egységet azonnal érvényesíthetünk
  - Ha egy egység egy másikra épül például egy hardverrel megvalósított de még nem kész egységre, akkor azt szimulálnunk kell

Spirál modell

- A programfejlesztési lépések mindegyike besorolható valamelyik dimenzióba, de a tevékenységek nem feltétlenül a dimenzió által meghatározott irányban követik egymást
- Például a rendszer dekomponálása során új elemeket (boxes) azonosíthatunk és ebben a pillanatban egy megtestesítés, egy dekompozíció, egy kompozíció és egy érvényesítési lépéssel állunk szemben
- Minimum a megtestesítést megtehetjük és a kompozíció és az érvényesítés lépéseit végrehajthatjuk az absztrakt szinten is, azaz az UML modell szintjén
- Ez azt jelenti számunkra, hogy a különböző szinteken ismétlődhetnek a fenti lépések – ez a spirál programfejlesztési modell lényege

2.1. ábra


Composition
Validation
Composition
Hardware
Source/Binary Code
Model
Concretization
Decomposition
Embodiment
Genericity
Abstraction
Specialization
Decomposition

Vízesés modell

- Vízesés vagy V-modellben a tevékenységek szigorúan egy meghatározott sorrendben történnek.
- Ezért a Spirál modell jobban megfelel a komponensalapú fejlesztésnek.
- A vízesés modell pedig a tradicionális szoftverfejlesztést szolgálja ki.

- A 2.3. ábrán bemutatjuk az árusító automata UML diagramját. Minden UML package szimbólum a modellelemek egy halmazát definiálja.

- Pénzbedobós automata:

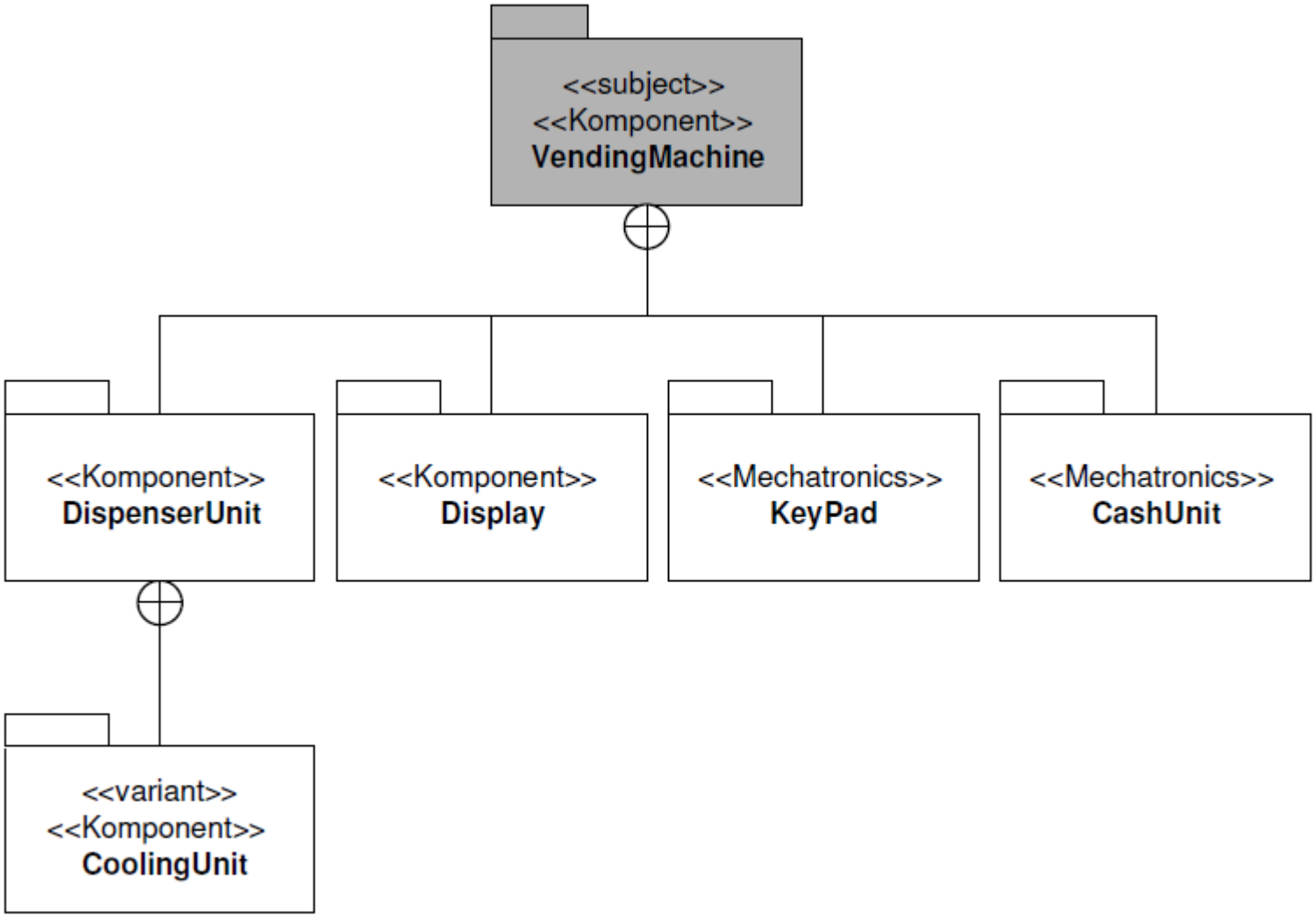


Fig. 2.3. Development-time containment tree for the vending machine example

Jelölések

- <<Komponent>>: jelöl egy KobrA komponenst
- <<Variant>>: jelöli azt a tényt, hogy a komponens egy termékcsalád része
- Mindenegyes <<Variant>> komponensnek más és más lehet az implementációja, attól függően, hogy milyen terméket akarunk létrehozni

Környezeti térkép

- A szoftverfejlesztés kiindulópontja a rendszer vagy alkalmazás **specifikációja**.
- A specifikációt a rendszerrel szemben támasztott követelményekből származtatjuk, dekomponáljuk. A felhasználói szintű követelményeket a szoftver leendő felhasználóitól gyűjtjük össze ***(use case diagrams*)** és ezután dekomponáljuk azokat, hasonlóan a készülő rendszer dekomponálásához.
- A környezet feltérképezése során a rendszer környezetéről ***leíró adatokat*** gyűjtünk össze.
- Elvárás, hogy a rendszer vagy alkalmazás specifikációja visszatükrözze annak a környezetnek az elvárásait, amelybe az elkészülő rendszert integráljuk.
- A környezetet definiáló leíróelemek összességét nevezzük **környezeti térképnek**.
- A környezeti térképet a KobrA módszer ugyanúgy kezeli mint a komponenseket. A környezet önálló, saját jogú komponens.
- A 2.4. ábrán az árusító automatánk környezeti térképének egy részletét mutatjuk be. A <<Mechatronics>> sztereotípia jelzi, hogy a rendszer elektronikus, mechanikus részek és szoftver kombinációjaként áll majd elő.

2.4. ábra

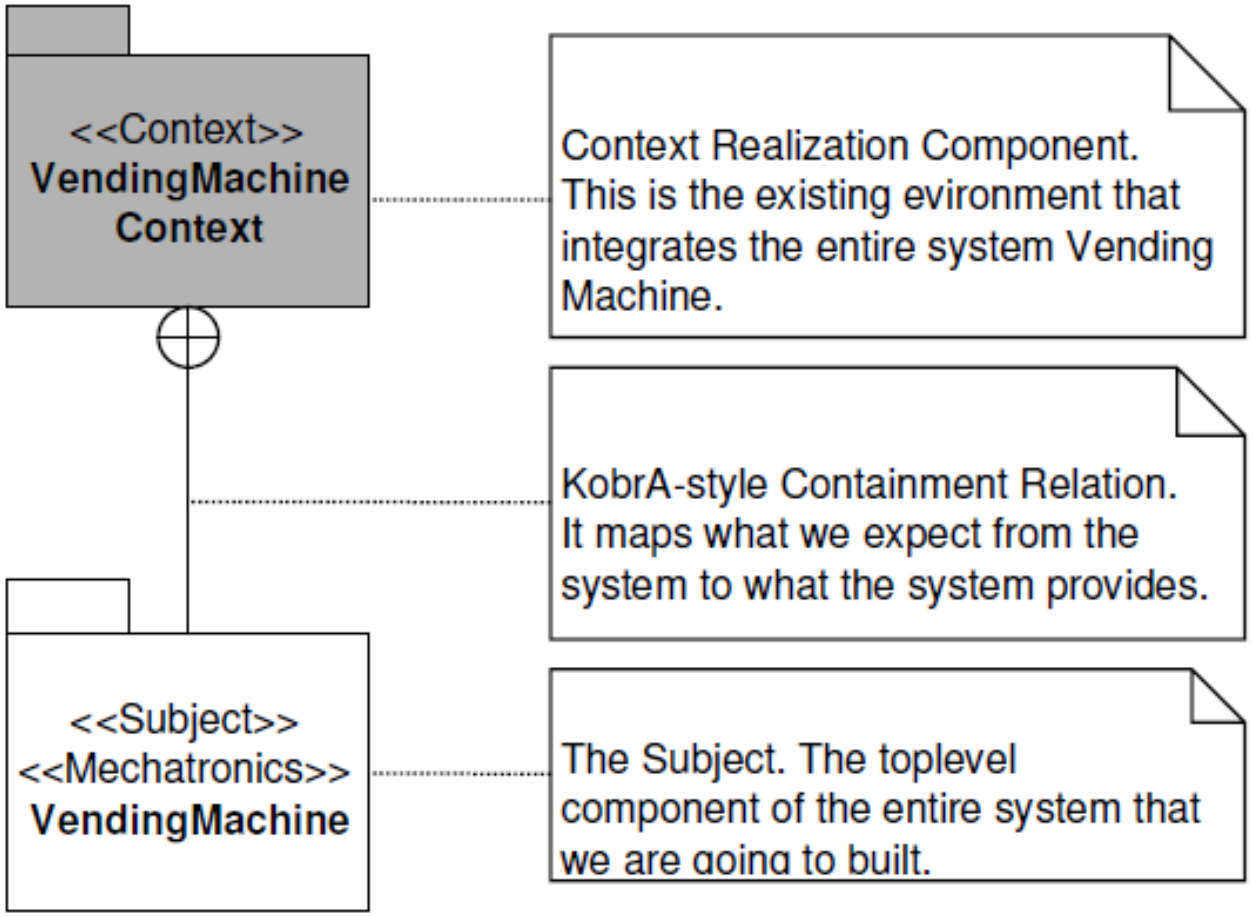


Fig. 2.4. Excerpt from the containment tree: vending machine context realization and vending machine component

- A környezeti térkép eredményeként tulajdonképpen az egész rendszerünknek az elvárt interfészét is megkapjuk.
- Ha találunk egy már kész komponenst, amelynek specifikációja pontosan kielégíti a kontextusunkat, akkor azt felhasználhatjuk, nem kell újra elkészítenünk.

- A környezet definiálása a fejlesztőprojekt első lépése, amelynek során a következő kérdésekre keressük a választ:
  - Milyen tevékenységet támogat a rendszerünk?
  - Mi a rendszerünk inputja?
  - Ki/mi fog kapcsolatba kerülni a rendszerünkkel?
  - Milyen objektumokra van szükség a rendszerből, hogy más aktivitást is végrehajtson a környezetben?
- Ezt az első lépést a használati esetek (use case diagrams) jól repzrezentálják.
- Környezeti térkép létrehozására további modellek is felhasználhatók: vállalati vagy üztleti folyamatok modellje, strukturális és kölcsönhatás modellek.
- A 2.5. ábra bemutatja azokat az elemeket, amelyek egy környezeti térképet alkotnak.

- 2.5 ábra

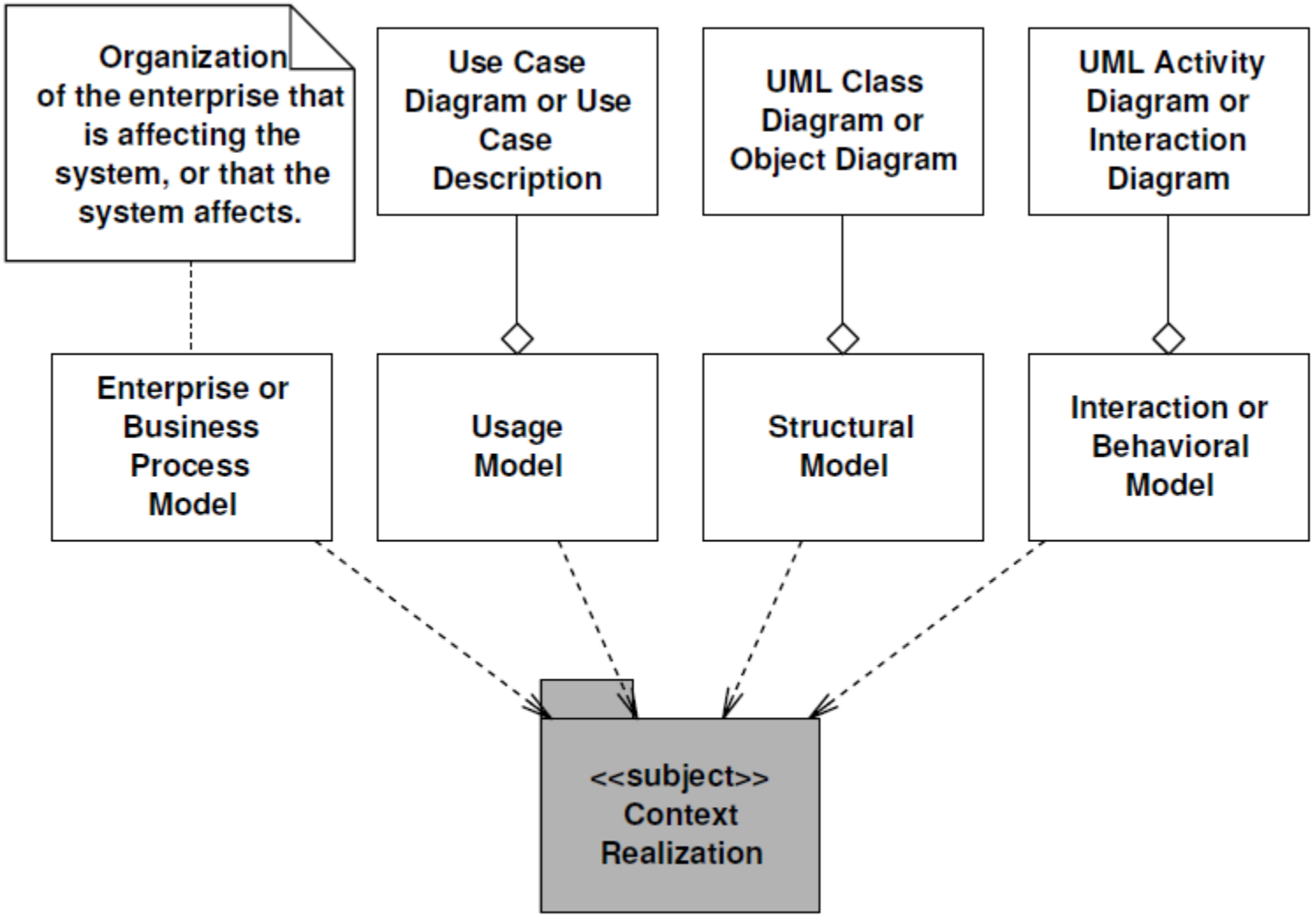


Fig. 2.5. UML-style definition of a generic context realization

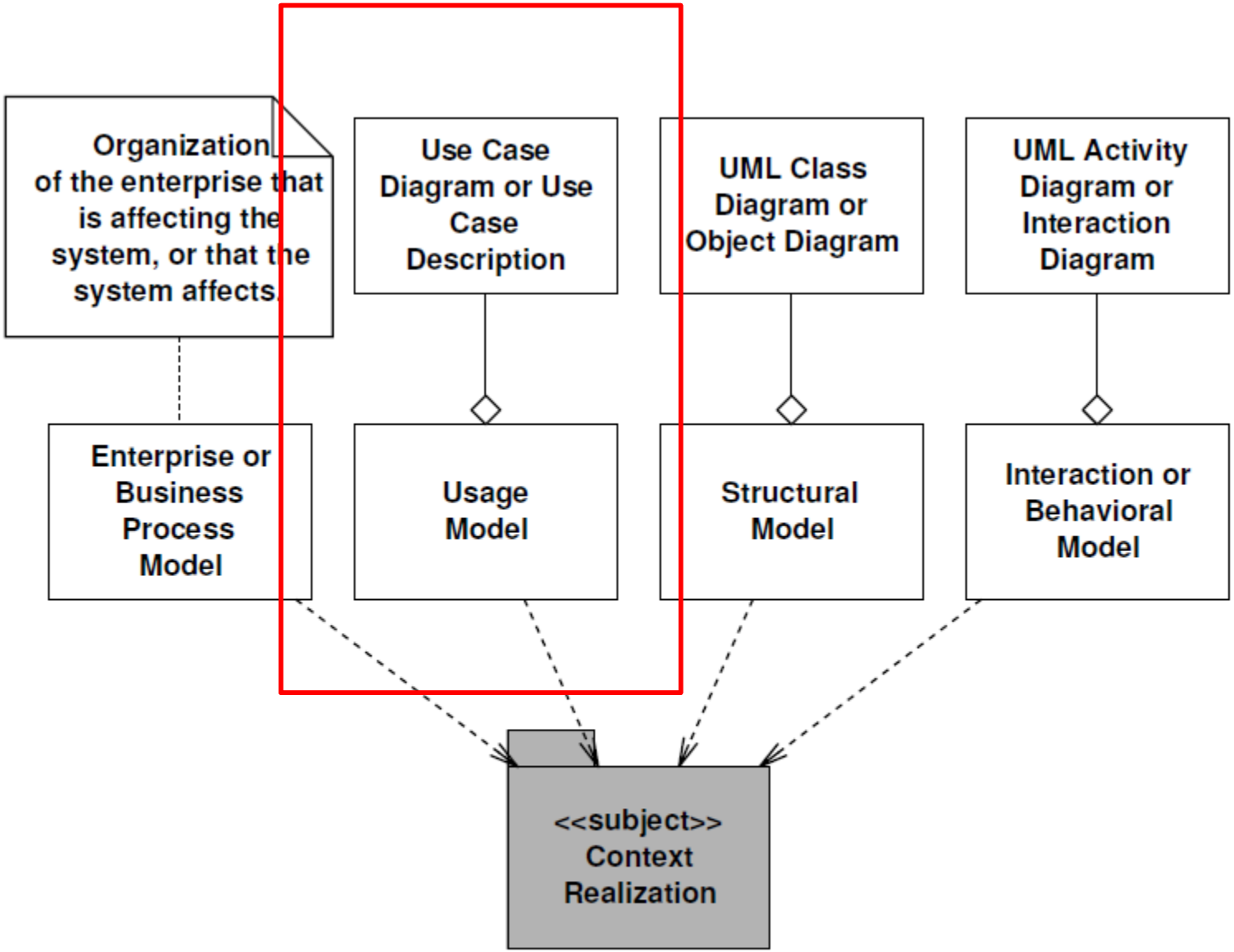


Fig. 2.5. UML-style definition of a generic context realization

Használati modell – Usage Model

- A felhasználónak a rendszerrel fenntartott magas szintű kapcsolatait specifikálja.
- A használati modell használati eset diagramok összességéből áll elő.
- A használati modellek arra valók, hogy a rendszermodulokat nagyon absztrakt szinten (durván szemcsézett módon) definiáljuk.
- A használati esetek főleg a rendszer összetevői közötti interakciókra, valamint a rendszer és határai közötti kapcsolatokra koncentrálnak.
- A használati eset diagramok bemutatják a rendszer szereplőit és a használati eseteket, valamint a közöttük létező kapcsolatokat.
- Minden használati eset valamilyen absztrakt tevékenységet reprezentál, amelyet a rendszer felhasználója hajt végre.
- A 2.6. ábra az árusító automata használati eset diagramját mutatja be.

- 2.6 ábra

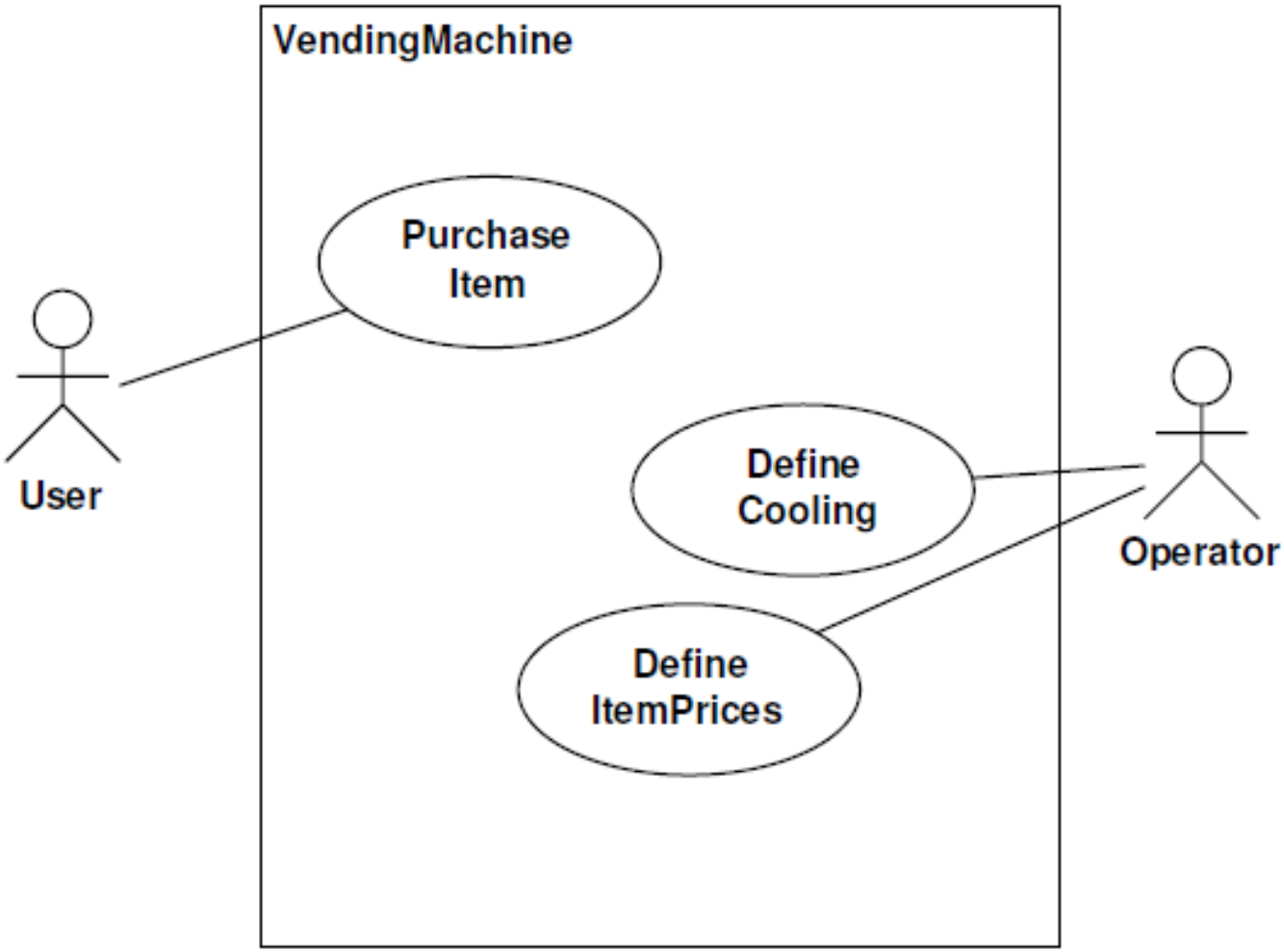


Fig. 2.6. Use case diagram for the vending machine

Használati modell (folyt.)

- A használati eset diagramok önmagukban nem sok információt hordoznak, ki kell őket egészíteni használati eset leírókkal. Erre a célra úgynevezett használati eset sablonokat (use case templates) használhatunk.
- A 2.1. táblázat egy ilyen sablont mutat be.

Table 2.1. Use case template

Use Case No.	Short name of the use case indicating its goal (from the model)
Goal in Context	Longer description of the use case in the context.
Scope & Level	Scope and level of the considered system, e.g. black box under design, summary, primary task, sub-function, etc.
Primary/Secondary Actors	Role name or description of the primary and secondary actors for the use case, people, or other associated systems.
Trigger	Which action of the primary/secondary actors initiate the use case.
Stakeholder & Interest	Name of the stakeholder and interest of the stakeholder in the use case.
Preconditions	Expected state of the system or its environment before the use case may be applied.
Postconditions on success	Expected state of the system or its environment after successful completion of the use case.
Postconditions on failure	Expected state of the system or its environment after unsuccessful completion of the use case.

folyt. köv.

Description Basic Course	Flow of events that are normally performed in the use case (numbered).
Description Alternative Courses	Flow of events that are performed in alternative scenarios (numbered).
Exceptions	Failure modes or deviations from the normal course.
NF-Requirements	Description of non-functional requirements (e.g. timing) according to the numbers of the basic/alternative courses.
Extensions	Associated use cases that extend the current use case (≪extends≫ relation).
Concurrent Uses	Use cases that can be applied concurrently to the current use case.
Revisions	Trace of the modifications of the current use case specification.

Használati modell (folyt. 2.)

- A 2.2. táblázat bemutatja a példa automatánk PurchaseItem használati esetének leírását.

Table 2.2. Definition of the use case *Purchase Item* from the vending machine usage model

Use Case 1	Purchase Item
Goal in Context	Main scenario for purchasing an item from the vending machine.
Actors	User.
Trigger	User inserts coins into slot, or selects item on button panel.
Preconditions	Vending machine is operational.
Postconditions on Success	Item is provided.
Postconditions on Failure	Item is not provided.
Description Basic Course	1. User inserts sufficient amount of money. 2. User selects item on panel. 3. Selected item and return money are dispensed. 4. User takes item and returned money, if applicable.
Description Alternative Courses	1. User selects item. 2. Vending machine displays price. 3. <basic course>

folyt. köv.

Exceptions	1. [insufficient amount] user inserts more cash, or aborts. 2. [selected item not available] user selects different item, or aborts.
NF-Requirements	1. Item should be dispensed not more than 3 seconds after selection. 2. After 3 sec of no user selection use case is aborted and inserted cash returned.
Extensions	<left open>
Concurrent Uses	<left open>
Revisions	<left open>

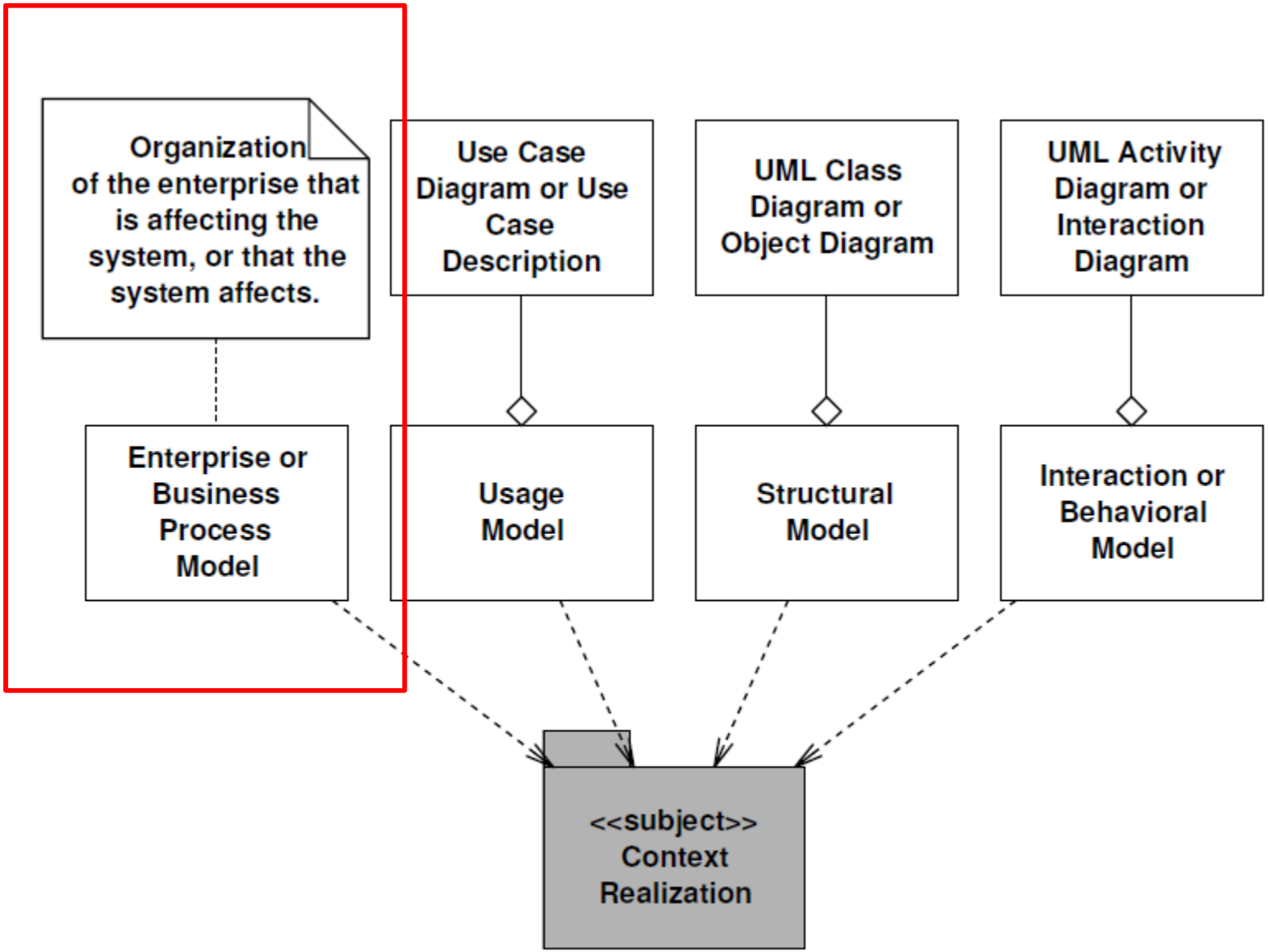


Fig. 2.5. UML-style definition of a generic context realization

Üzleti & vállalati folyamatok modellje

- A rendszerünk üzleti környezetét írja le.
- Néha ez kevésbé fontos, például a beágyazott rendszerek esetében (pénzbedobós automata).
- Néha fontos (banki rendszer).
- Banki rendszerek esetében a vállalati modell reprezentálja azokat a fogalmakat, amelyek a banki világ számára relevánsak.
  - A vállalati és üzleti modell ebben az esetben arra koncentrál, hogy ezek a létező fogalmak hogyan hatnak a számítógépes rendszerre.

Egy banki szoftver üzleti modellje

- Egy banki szoftver esetében releváns fogalmak:
  - Folyószámla
  - Számla
  - Átváltási ráták
  - Az ügyfél által kitöltendő űrlapok
  - …
- A vállalati és üzleti modellben azt kell megadni,
  - hogy ezek a fogalmak milyen hatással lesznek az elkészülő szoftverrendszerre és
  - a rendszer installálása után milyen változtatásokat kell eszközölni a bank szervezeti struktúrájában.

- 2.5 ábra

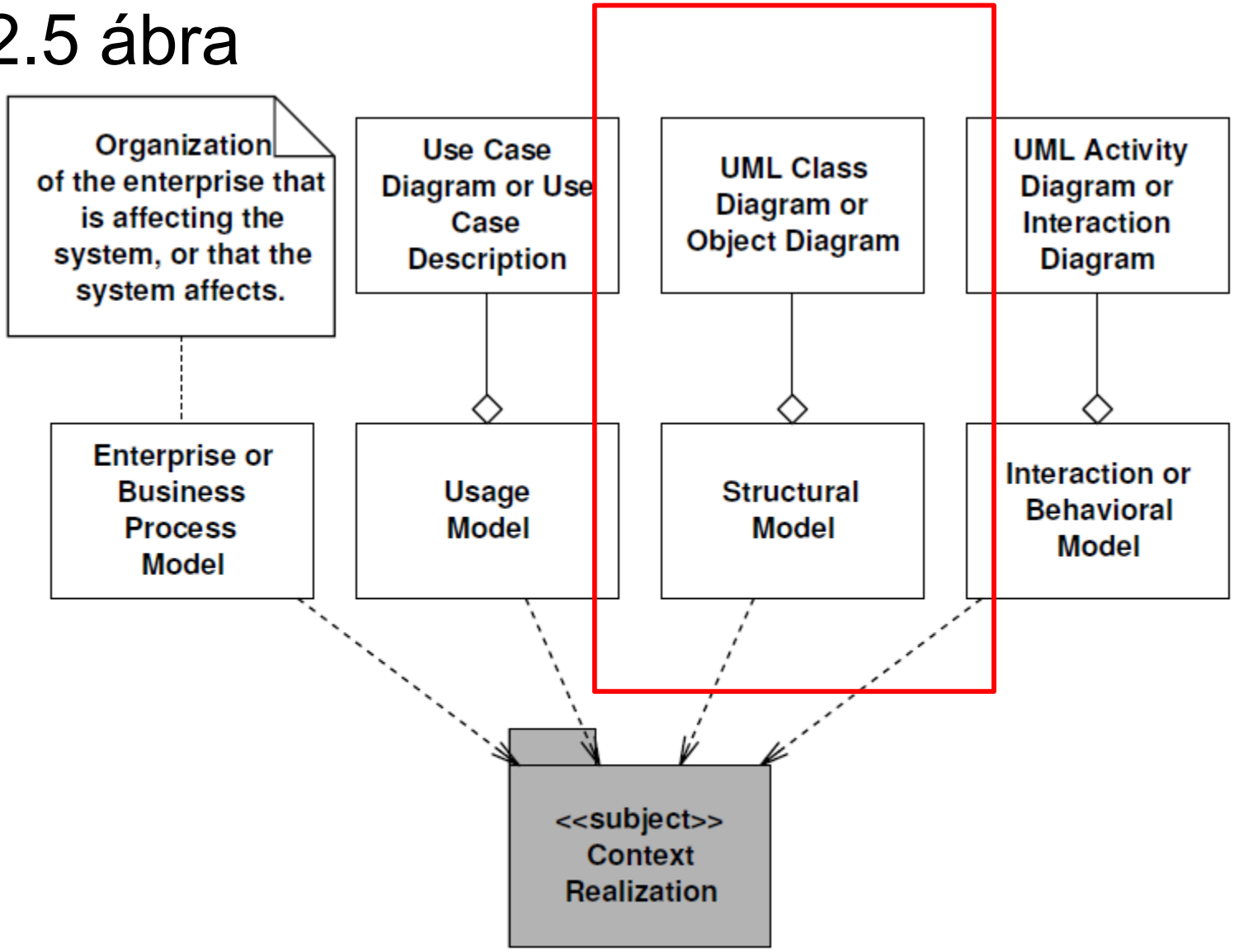


Fig. 2.5. UML-style definition of a generic context realization

Strukturális modell

- A környezeti térképen belüli strukturális modell definiálja azon entitások struktúráját, amelyek kívül esnek a a kifejlesztendő rendszer hatáskörén, de hatással vannak a rendszerre. Ez felöleli a rendszer
  - inputját és az olyan
  - outputját, amelyet valamilyen módon tovább feldolgoznak.
- A példánkban a legegyszerűbb esetben nincs olyan külső struktúra, amelyet figyelembe kellene venni, de ha például lenne bankkártyás vagy hotelkártyás fizetésre mód, akkor ezeket fel kellene venni a strukturális modellbe.
- A 2.7. ábra egy ilyen variáns környezeti térképen belüli strukturális modelljét mutatja be.

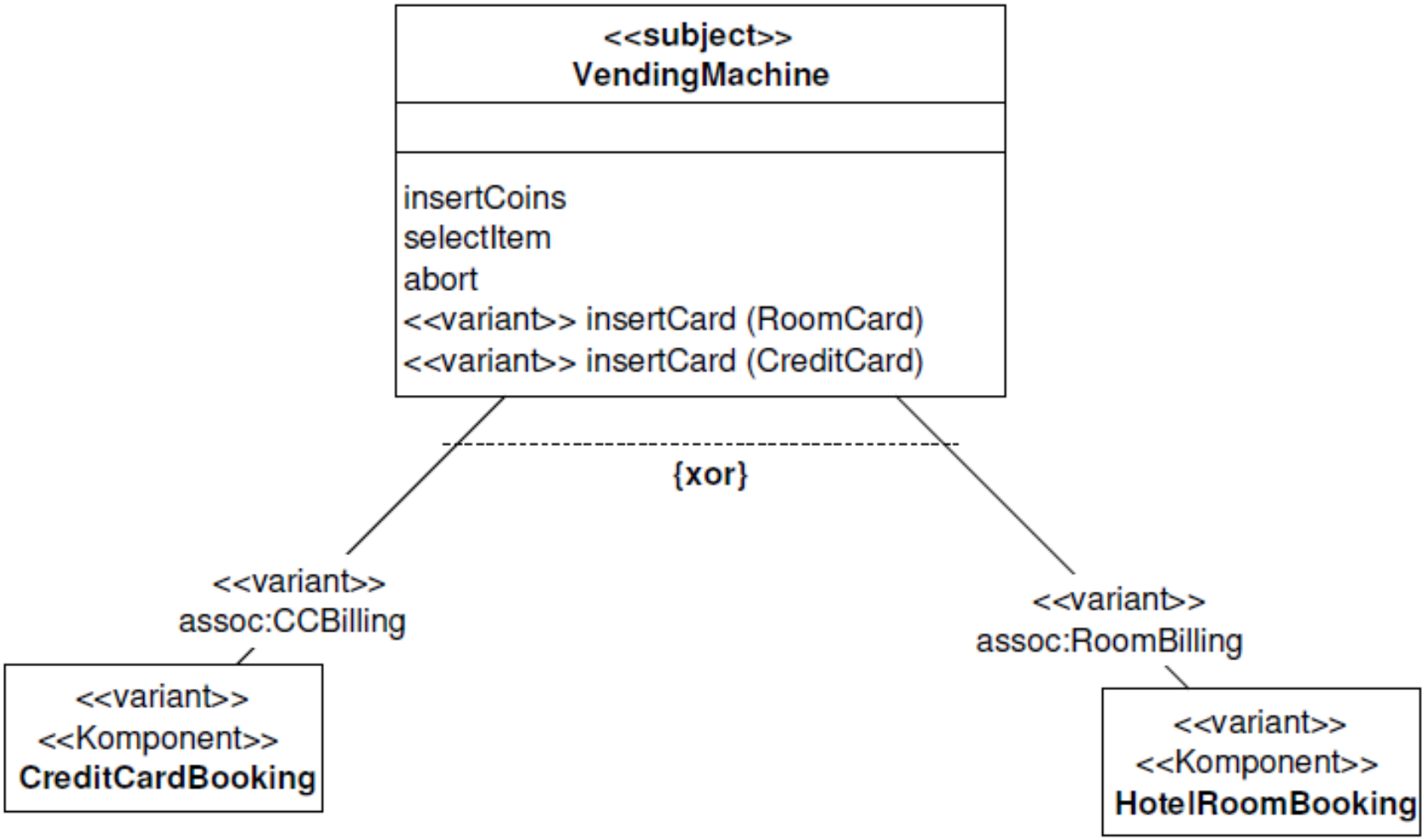
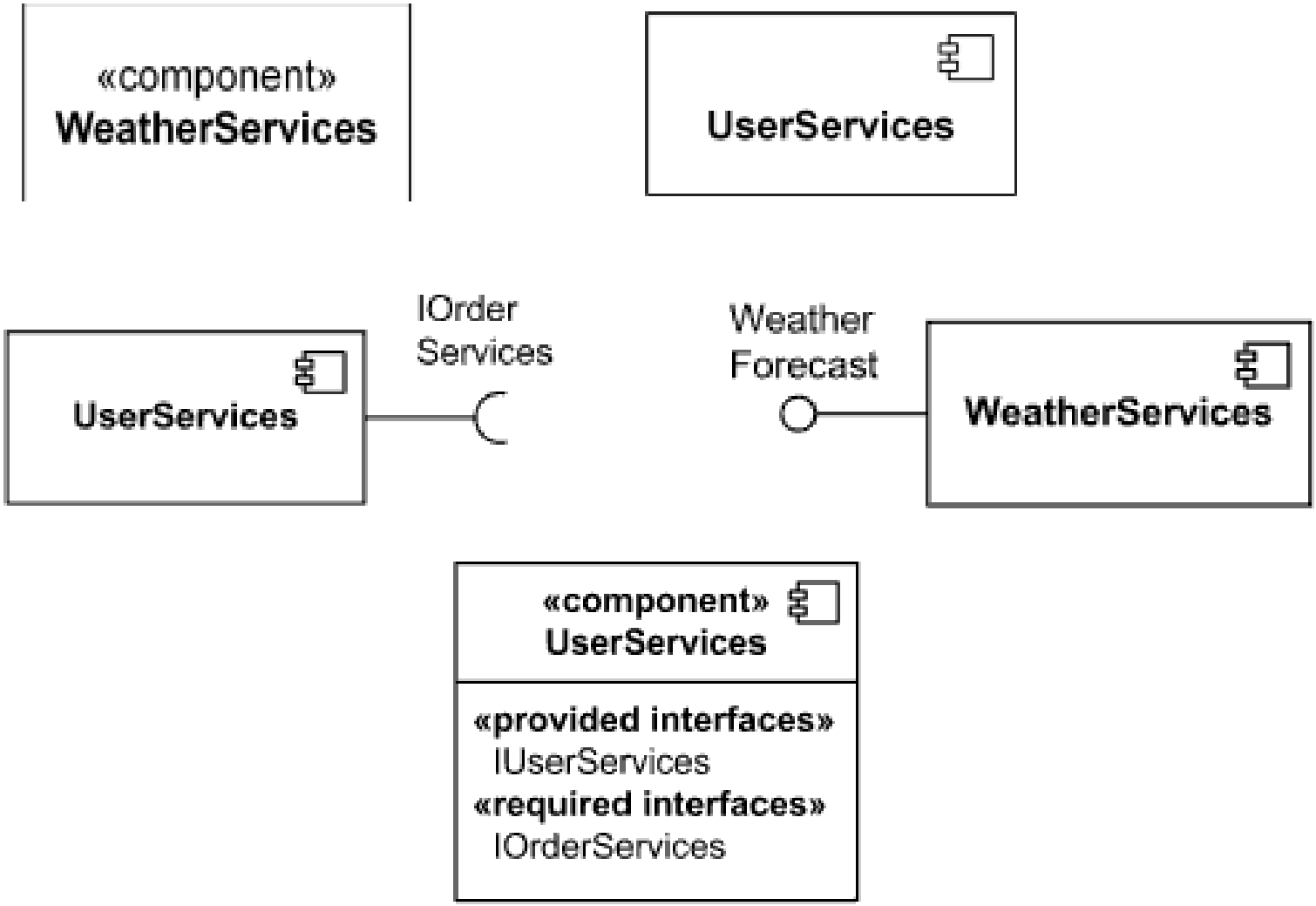
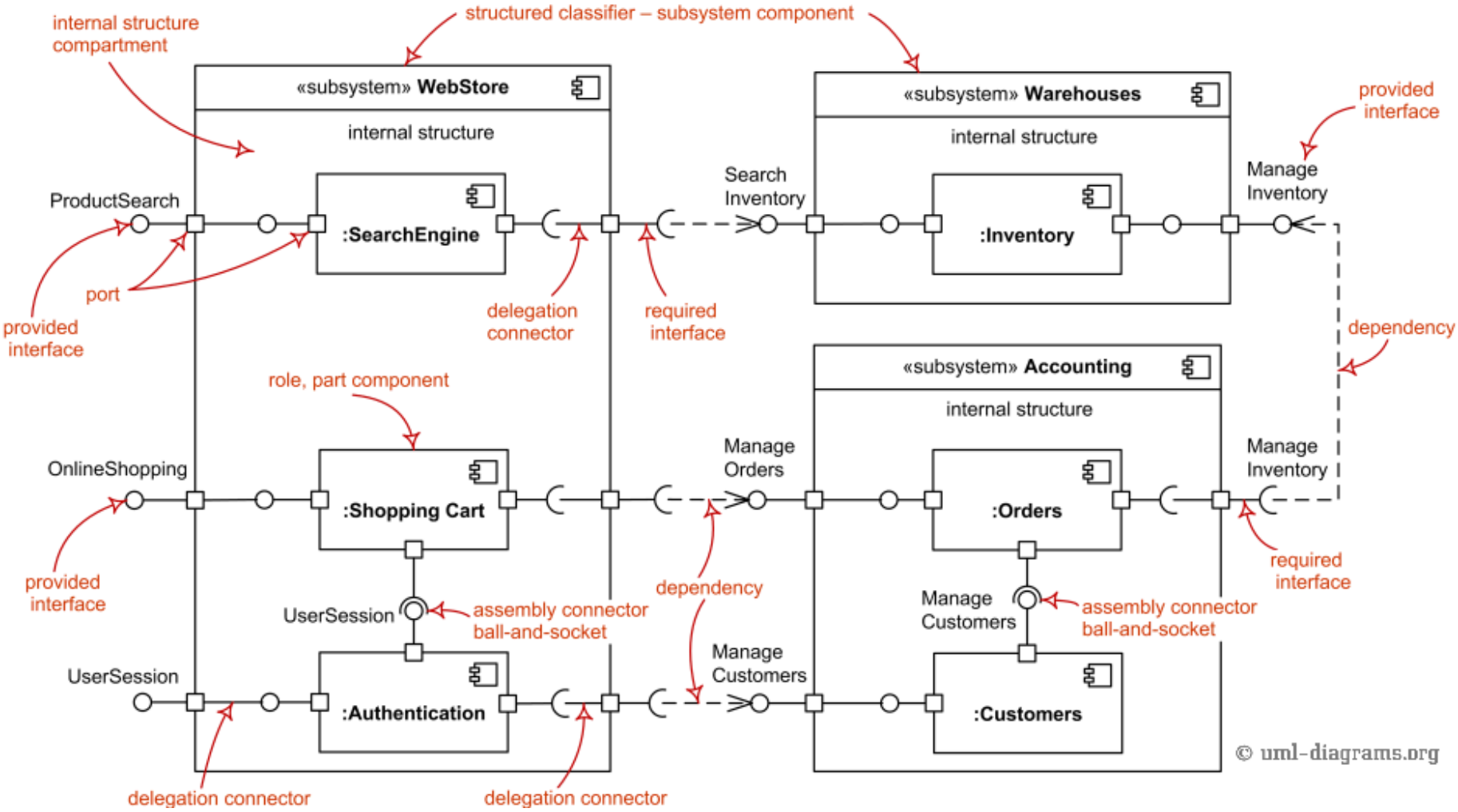


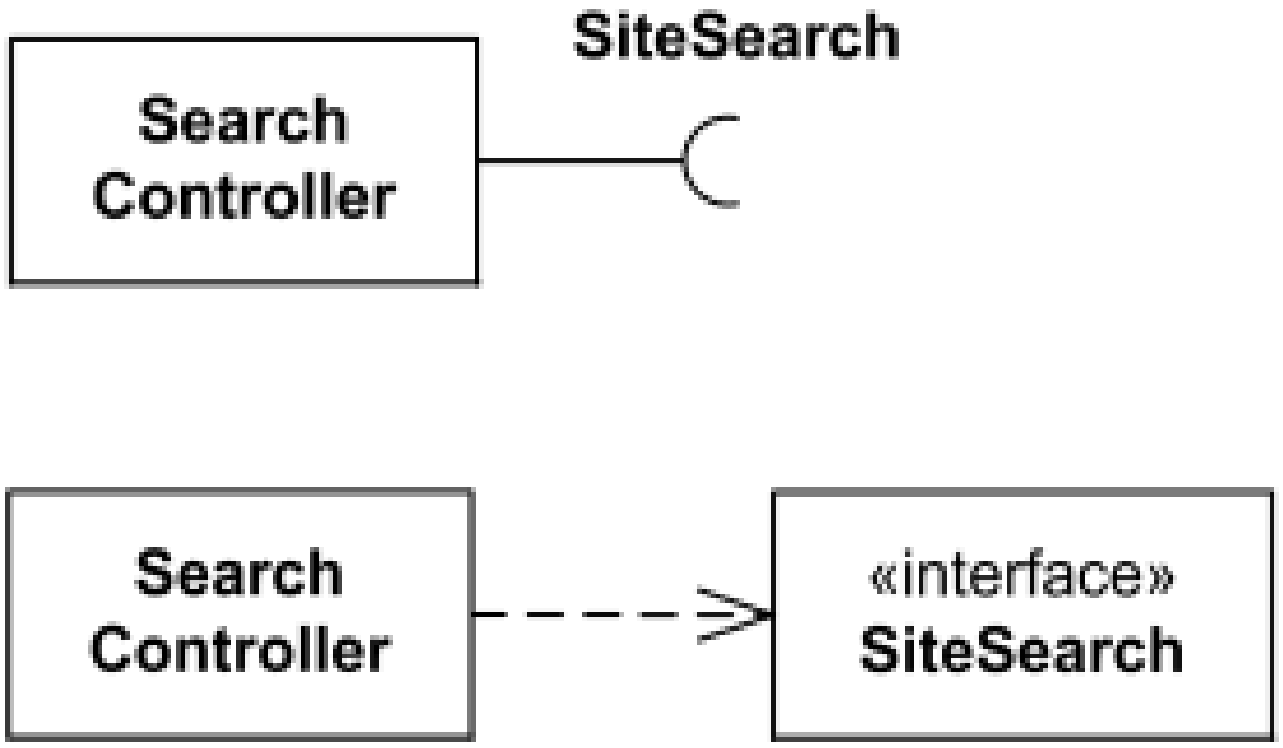
Fig. 2.7. Context realization structural model for the vending machine context

Jelölések

- Az árusító automata felhasználója ebben az esetben háromféle módon fizethet:
  - a hotel számlájával együtt, vagy
  - a bankkártyájával, vagy
  - pénzérmével.
- Ezt jelezzük a változatok segítségével:
  - << variant >> RoomBilling és << variant >> CCBilling asszociáció a diagramban illetve
  - << variant >> insertCard(…) műveletek a diagramban
  - a harmadik az alapeset.


«component»
WeatherServices
UserServices
UserServices
IOrder
Services
Weather
Forecast
WeatherServices
«component»
UserServices
«provided interfaces»
IUserServices
«required interfaces»
IOrderServices


internal structure compartment
structured classifier – subsystem component
«subsystem» WebStore
internal structure
«subsystem» Warehouses
internal structure
provided interface
ProductSearch
:SearchEngine
Search Inventory
:Inventory
Manage Inventory
port
provided interface
delegation connector
required interface
dependency
«subsystem» Accounting
internal structure
role, part component
OnlineShopping
:Shopping Cart
Manage Orders
:Orders
Manage Inventory
provided interface
UserSession
assembly connector ball-and-socket
dependency
Manage Customers
assembly connector ball-and-socket
required interface
UserSession
:Authentication
Manage Customers
:Customers
delegation connector
delegation connector
© uml-diagrams.org


SiteSearch
Search Controller
Search Controller
«interface»
SiteSearch

- 2.5 ábra

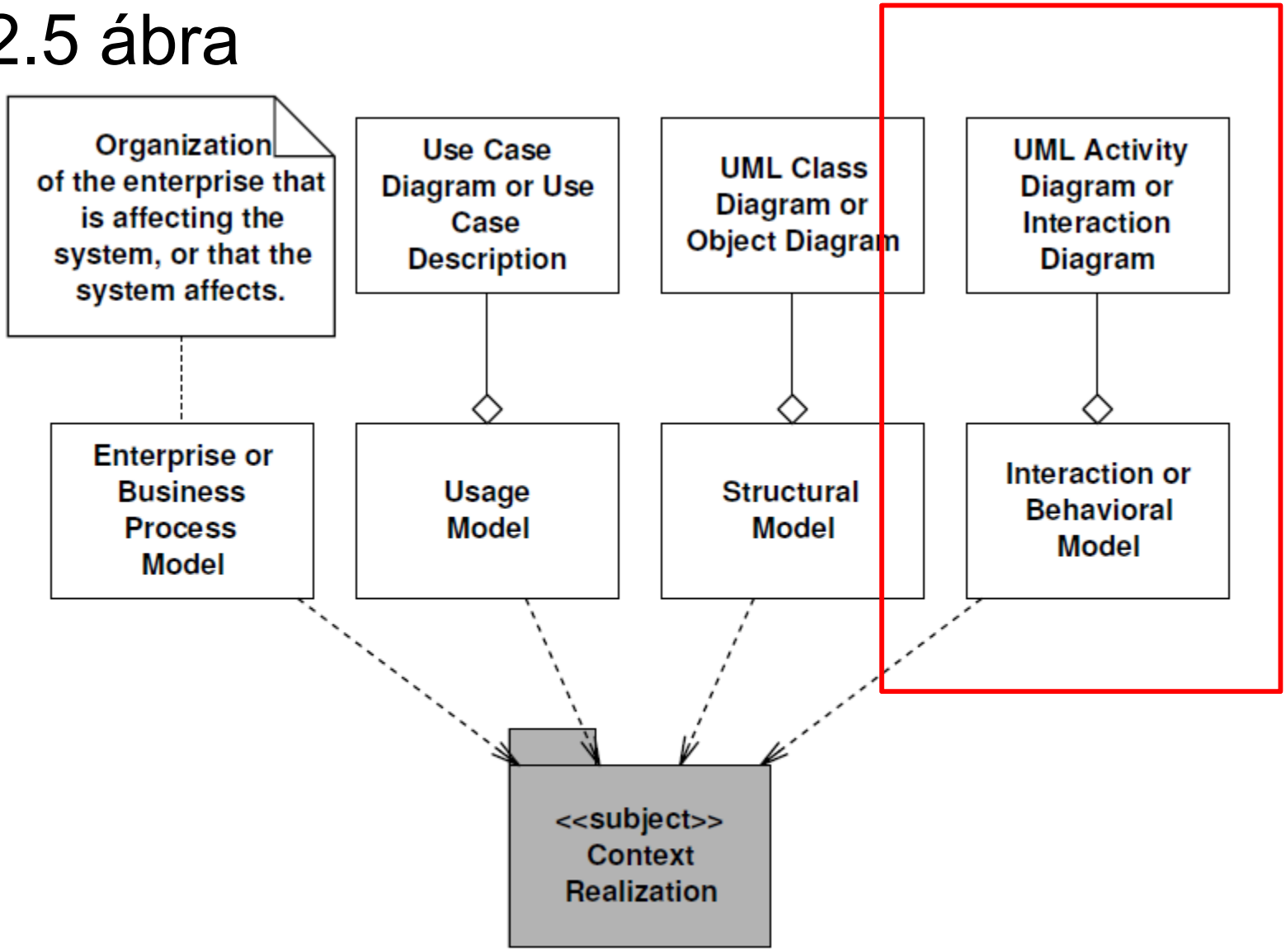


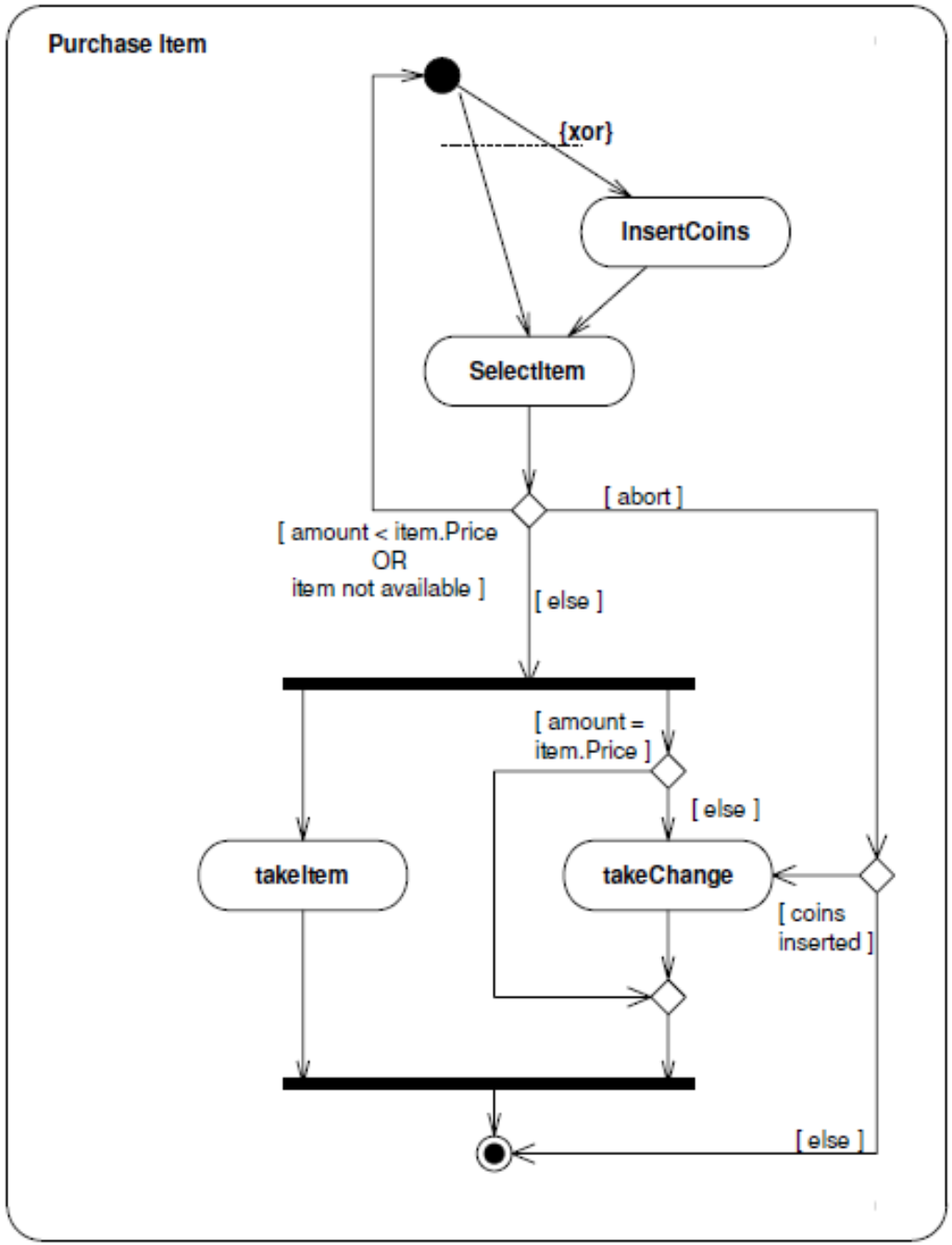
Fig. 2.5. UML-style definition of a generic context realization

Interakciós és viselkedési modell

- A használati eset modell
  - alapján megvitathatjuk a megrendelővel, hogy jól megértettük-e feladatot.
  - az elkészült modelleket diagramokká transzformáljuk, képezzük le.
- Az interakciós és viselkedés modell
  - alapján készült diagramok technikai részletek absztrakciója felé tesz lépéseket, de nem túl nagyokat. A kis lépések stratégiája nagyon fontos. A rendszer kifejlesztése rekurzív módon történik, amíg olyan változathoz érünk el, amely már megvalósítható.

Interakciós és viselkedési modell

- A környezeti térképen belüli **viselkedési modell** a rendszer szereplőinek a rendszer határára vonatkozó tevékenységeit írja le.
- Az UML aktivitás diagramjával történik
- nagyon hasonlít a tradicionális kontrol folyam gráfra, bár a diagramokat magasabb absztrakciós szinten használják.
- A 2.8.ábra egy olyan aktivitás diagramot mutat be, amelyet a 2.2. tábla (az árusító automata *Purchase Item* használati eset leírója) alapján készítettünk el.

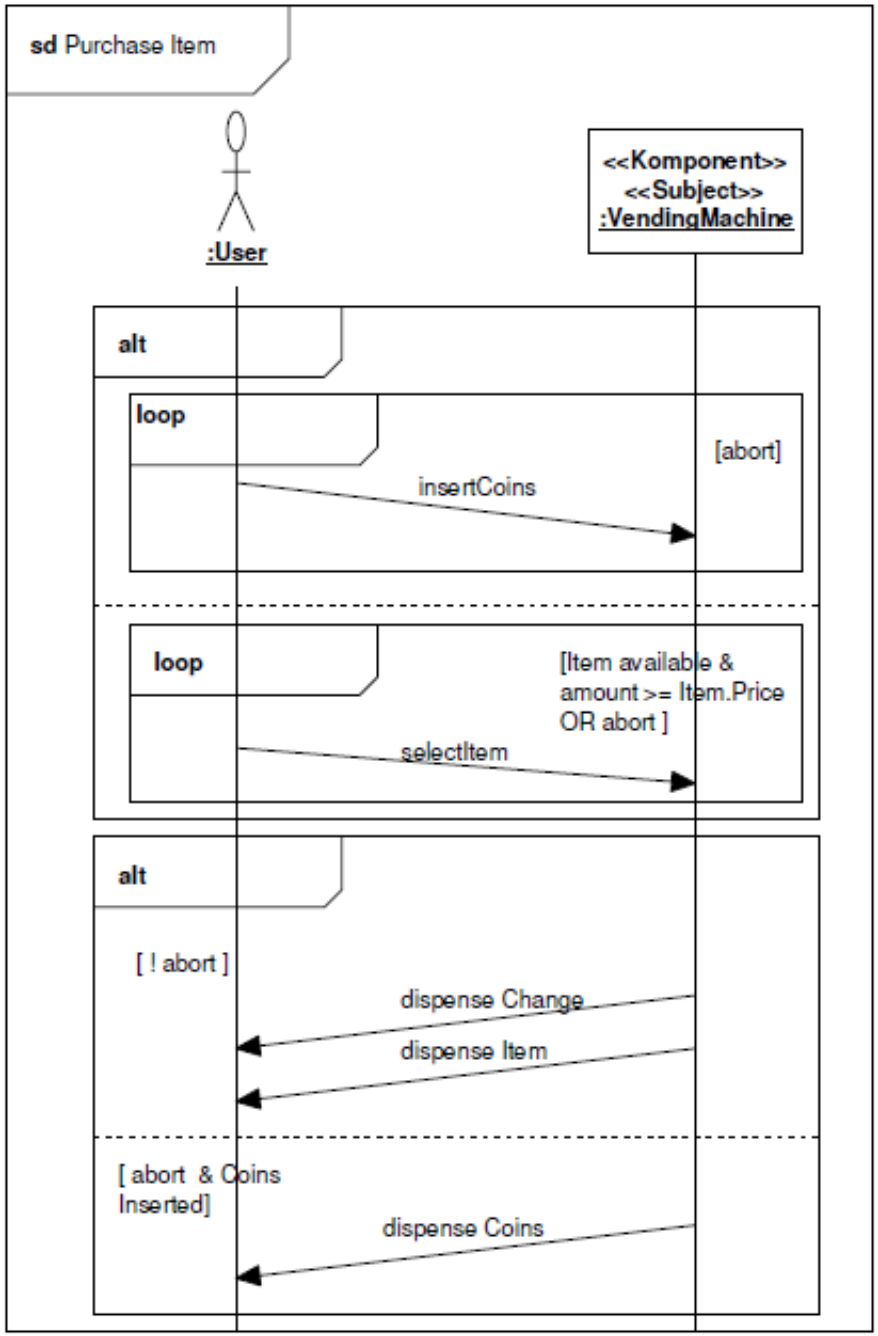

Purchase Item
{xor}
InsertCoins
SelectItem
[abort]
[amount < item.Price
OR
item not available]
[else]
[amount =
item.Price]
[else]
takeItem
takeChange
[coins
inserted]
[else]

Magyarázat

- A diagram belépési pontján két alternatíva közül választhatunk
  - választunk egy megvásárolandó tételt vagy
  - először bedobjuk a pénzt és csak azután választunk tételt.
- Az aktivitási diagramban a fő haladási irány a kiválasztott tétel kiadása az esetlegesen visszajáró pénzzel együtt.
  - E két tevékenység folyhat párhuzamosan, ahogy a modellből leolvasható.
- A másik lehetséges haladási irány az abortálás.

Interakciós vagy viselkedési modell

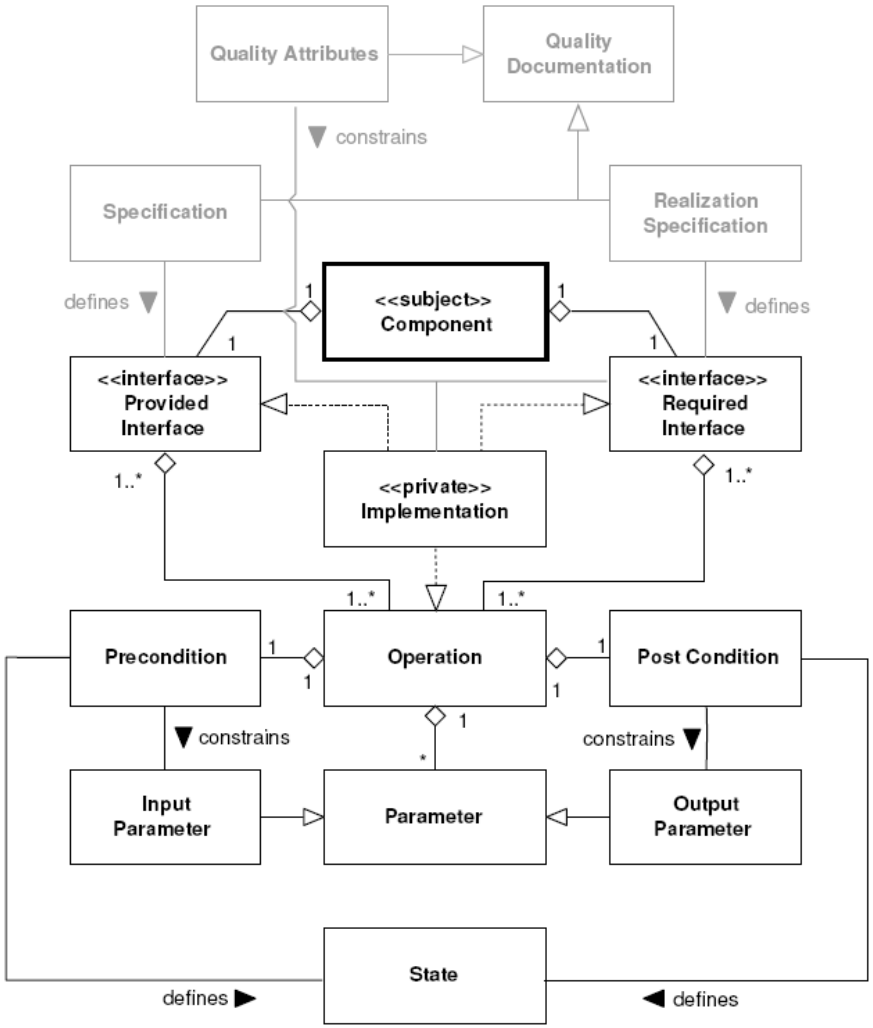
- Az interakciós (kölcsönhatás) modell
  - nagyon hasznos a felhasználó aktivitásának megjelenítése szempontjából.
  - A 2.9. ábra a példa automatánk felhasználójának cselekedeteit mutatja be.


sd Purchase Item
:User
<<Komponent>>
<<Subject>>
:VendingMachine
alt
loop
[abort]
insertCoins
loop
[Item available &
amount >= Item.Price
OR abort]
selectItem
alt
[! abort]
dispense Change
dispense Item
[abort & Coins
Inserted]
dispense Coins

A környezeti térkép összefoglalása

- Ha a környezet térkép elkészült, akkor a következő lépés, hogy a legfelső szinten lévő komponenst, azaz a rendszert specifikáljuk.
- Komponens/rendszer: nézőpont kérdése, nehéz az elválasztó határvonalat meghúzni.
- Az egyes komponenseket az 1.3. ábrán bemutatott meta modell eszközeivel definiálhatjuk (lásd következő dián is.)

Komponens meta-modell diagram



- A KobrA módszer következő két fontos eleme:
  - A komponens specifikáció és
  - A komponens megvalósítás.
- A továbbiakban ezekkel a kérdésekkel foglalkozunk.