- eddig volt:
  - mi a komponens
  - mi a komponens modell
  - mik a komponens modell részei
  - ismert hogyan írhatunk le komponensalapú rendszereket UML segítségével
  - módszer komponensalapú rendszerek tervezéséhez:
    - háromdimenziós modell, környezeti térkép
  - módszer komponensalapú rendszerek tervezéséhez
    - komponensek, specifikáció, realizáció
- most jön:
  - komponensek újrafelhasználása, termékcsalád

Komponens megtestesítés

- A **komponens megtestesítés** mindazon tevékenységek összefoglaló neve, amelyek során a rendszerünk absztrakt specifikációból konkrétabb, például végrehajtható, vagy telepíthető komponenst hozunk létre.
- A KobrA módszer esetében a létrehozandó rendszert a legfelső absztrakciós szinten egyetlen komponensnek tekintjük.
  - Ezt a legmagasabb szintű komponenst alacsonyabb szintű komponensekké dekomponáljuk a dekompozíciós és a konkretizációs dimenziók mentén.
  - Végezetül létrehozzuk a futtatható, bináris kódot.
- Ezt a folyamatot illusztrálja a 2.19. ábra. Ahogy már említettük a szoftverfejlesztés fenti lépései a vízesés modellel ellentétben nem egymás utáni követik egymást, hanem spirálszerűen.

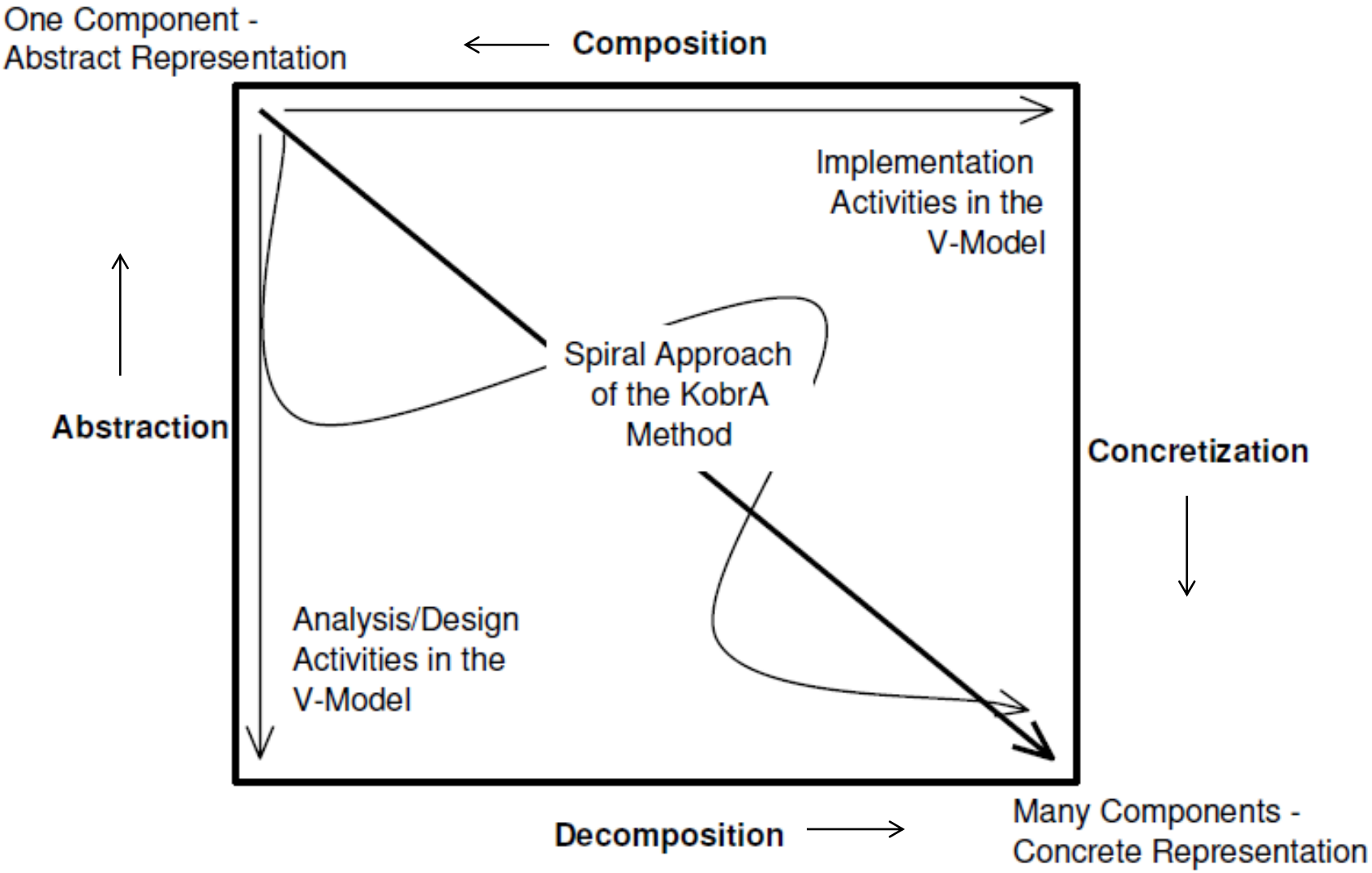

Fig. 2.19. Spiral approach of the KobrA method vs. approach of the waterfall/V-model

- A fenti folyamat egyes lépései automatizálhatók, például a magas szintű programozási nyelven elkészült forráskódot automatikusan bináris kóddá transzformálja a fordítóprogram.
- Egy modell transzformálása forráskódú programmá azonban tipikusan olyan tevékenység, amely emberi, manuális tevékenységet igényel, mivel a modell és a forráskódú program szemantikus leírása között egy jelentős ugrás van, ugyanis a jelölésmód és annak szemantikája is változik.
  - A transzformáció
    - inputja: UML diagramok
    - outputja : programsorok.
- Ezt a transzformációt illusztrálja a 2.20. ábra.

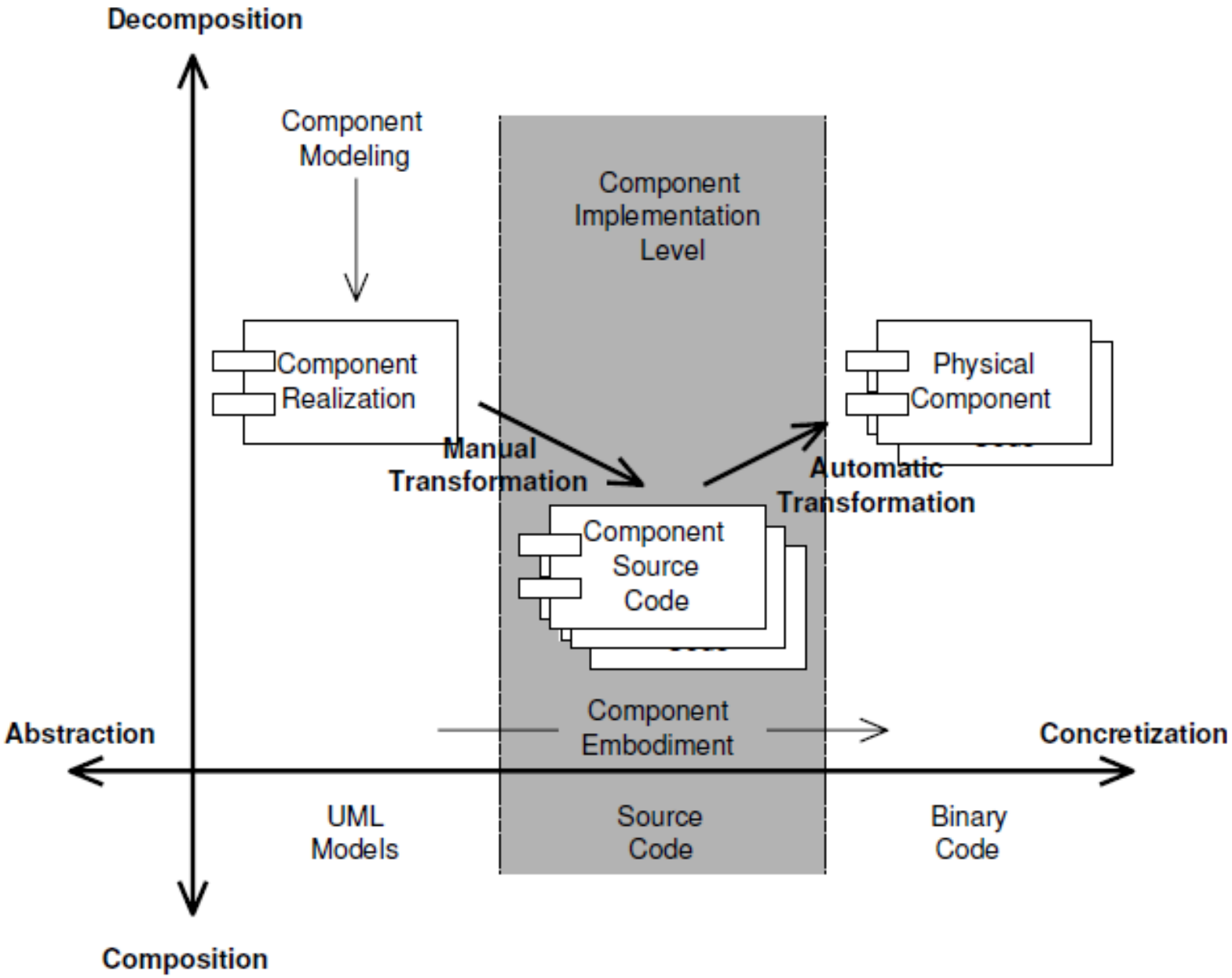

Fig. 2.20. Refinement and translation (Manual Transformation) in a single step

- Ezt az információs szakadékot úgy tudjuk áthidalni, hogy a modellben használt fogalmakat transzformáljuk a megvalósítás célnyelvének választott programozási nyelv megfelelő fogalmaivá.
  - Az OO nyelveknél ez egyszerűbb feladat, mert ezek támogatják a legtöbb UML fogalmat.
  - Tradicionális procedurális nyelvek esetében ezt a szemantikai szakadékot sokkal nehezebb áthidalni.

- A fenti manuális transzformáció során gyakran kell programozási nyelvi szinten olyan implementációs döntéseket hoznunk, amelyek eredményeként a szoftver és a modell eltérnek egymástól.
  - Ez a tény aláássa a modell alapú megközelítés létjogosultságát. Ennek oka az, hogy a tervezést a legmagasabb szinten végezzük, amely egy nagyon durva nézetét adja az egész rendszernek és az implementációs megfontolásokat közvetlenül ezen a nézet alapján hozzuk meg.
  - Tovább rontja a helyzetet, hogy ezek a döntések általában nem kerülnek dokumentálásra a magasabb szintű modellekben.
  - A továbbiakban azt mutatjuk be, hogy a fenti probléma hogyan hidalható át.

SORT technika

- A KobrA módszer használói elsősorban a **Systematic Object-Oriented Refinement and Translation** (SORT) technikát alkalmazzák a megtestesítési folyamat ezen fázisában.
- A SORT technika két alapvető elven nyugszik:
  - szigorúan szeparáljuk és különböztessük meg egymástól a finomítást és a fordítást;
  - használjuk a Normal Object Form (NOF) implementációs profilt, hogy minimalizáljuk az információs szakadékot az objektum-orientált modellek és az adott objektum-orientált nyelven elkészítendő program között.

Finomítás és fordítás

- A **finomítás e**gy reláció ugyanazon dolog kétféle leírása között, pontosabban fogalmazva a finomítás ugyanazon dolog jobban részletezett leírása egy másik szinten.
- A **fordítás** egy reláció két különböző leírás között a részletezettség azonos szintjén.
- A két különböző megközelítés keverése a szoftverfejlesztési projektben a rendszer olyan bonyolult reprezentációjához vezet, amelyet nehéz megérteni. Legrosszabb esetben egy olyan rendszerhez jutunk, ami egyáltalán nem felel meg a modellnek.
- Következésképpen a finomítást és a fordítást elkülönülten, mint egyedi aktivitásokat kell kezelni, ahogy ezt a 2.21. ábra bemutatja.

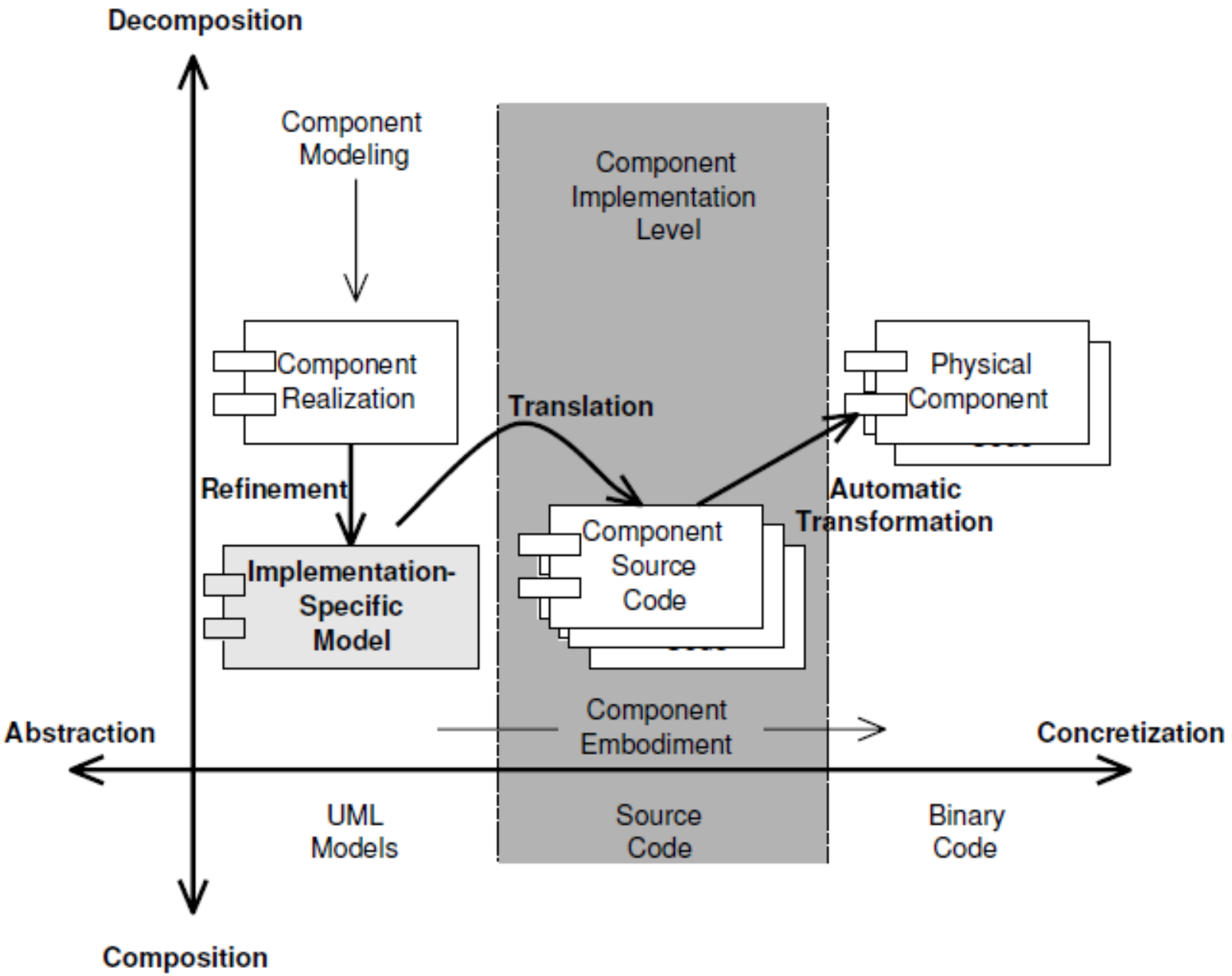

Fig. 2.21. Refinement and translation in two separated steps

- A 2.21. ábrán követett filozófia az, hogy a létező modelljeinket **első lépésben** finomítsuk egy előre definiált szintű modellekké majd a **második lépésben** ezeket a modelleket transzformáljuk forráskóddá.
  - Itt az a probléma vetődik fel, hogy hogyan definiáljuk azt a megfelelő részletezettségű szintet, amely már alkalmas arra, hogy onnan transzformációval megkapjuk a forráskódot.
  - Ez a probléma az implementációs minták segítségével megoldható.
    - Egy ilyen implementációs minta a Normal Object Form (NOF).

Normal Object Form

- UML profilok definiálhatók számos motiváció alapján
  - Tesztelési profil
  - Implementációs profil
    - Például a Java implementációs profil a Java standard nyelvi elemeket írja le, ragadja meg UML-ben
  - …
- A NOF egy ilyen implementációs profil, amely az UML-t képezi le az OO nyelvek alapvető (mag) fogalmaira. Ez a magleírás aztán tovább finomítható az egyes programozási nyelvekre, például az Eiffelre.

A NOF alkotóelemei

- Egy olyan UML részhalmaz, amely az implementációs szint elemeinek modellezésére alkalmas
- Olyan új modellező elemek, amelyek UML sztereotípiákon alapulnak
- Megszorítások a létező és az új modellező elemekre
- A NOF modellek érvényes UML modellek, a különbség abban áll, hogy az UML modellek csak az OO nyelvi koncepciókhoz állnak közel, míg a NOF az OO nyelven megírt programok modell alapú tervezését is támogatja.

A NOF használata

- A NOF modellek az OO programok olyan grafikus reprezentációját teszi lehetővé, amelyek a forráskódhoz közeli formájúak, így automatikusan transzformálhatók az adott programozási nyelvre.
- A NOF nagyon jól támogatja az Object Management Group modell vezérelt architektúrán alapuló szoftverfejlesztési koncepcióját.
- **Irodalom**
- C. Bunse & C. Atkinson. The normal object form: Bridging the gap from models to code. In 2nd International Conference on the UML, Fort Collins, USA,1999.

A SORT előnyei

- A Systematic Object-Oriented Refinement and Translation (SORT) technika alkalmazása nagyban megkönnyíti a megtestesítés folyamatát, ha betartjuk a következőket:
  - kis lépésekkel haladjunk előre
    - a kisebb lépések sorozatát könnyebb megérteni mint egy nagy, összetett lépését
    - a SORT technika éppen azt teszi lehetővé, hogy a grafikus modellek implementációját feldaraboljuk finomítási és fordítási lépések sorozatára

A SORT előnyei 2

- szeparáljuk a „gondokat”
  - a fejlesztők koncentráljanak egyszerre egyetlen tevékenységre
  - ezáltal a fordítást megelőző finomítási lépéseket egyszerűbb megtalálni
- azonosítsuk és használjuk ki a hasonlóságokat
  - a SORT támogatja a többféle implementáció létrehozását
  - általában egy új implementáció megalkotása nehéz feladat
    - a SORT ezt a nehézséget jelentős mértékben csökkenti, mivel a fejlesztőknek csak újra kell fordítani a komponens korábban létrehozott, NOF formában létező modelljét.

Komponensek újrafelhasználása

Komponensek újrafelhasználása

- A komponens újrafelhasználás mind a kompozíció/dekompozícó mind pedig az absztrakció/konkretizáció dimenziókhoz tartozó aktivitásokat reprezentál.
- Általában az újrafelhasznált komponens nem pontosan felel meg a specifikációnknak.

- Megoldási lehetőségek:
  - Az elkészített modell megváltoztatása, hogy így tüntessük el a különbségeket
    - Nem jó, szembemegy a komponensalapú fejlesztés elveivel
  - Az újrafelhasznált komponens átírása
    - Nem jó, ez túl bonyolult, időnként lehetetlen.
  - Átalakító (adapter) komponens bevezetése, ami leképzi egymásra a két interfészt
    - Wrapperosztályok használata
      - Számos komponens technológia szolgáltatásai között szerepelnek úgynevezett szintaktikus leképezések

Szemantikus leképezések

- Az egymással kapcsolatban álló entitások közötti szemantikus leképezések jelentik az igazi megoldást.
- A szemantikus leképezések lefordítják az egyik komponens „gondolatait” olyan formára, amelyet a másik „megért”.
  - Egy ilyen wrapper komponens neve „glue code” , amely általában a szerver komponenst zárja egységbe.

Új komponens elkészítése

- Ha nem találunk megfelelő komponenst, akkor magunknak kell elkészíteni.
- Ebben az esetben további dekompozíciós lépések után megtalálhatjuk az illeszkedő implementációkat.
- Ezután visszaléphetünk újra az integrációs lépéshez, hogy az újonnan implementált komponenseket a rendszerünkhöz illesszük.

Példa a **CashUnit**

- Tegyük fel, hogy az árusító automatánkhoz nem találunk a piacon nekünk megfelelő olyan kereskedelmi komponenst, amely az elvárásainknak megfelelően működő CashUnit szerepét tölthetné be.
- Ebben az esetben azt vizsgáljuk meg, hogy találunk-e a piacon olyan komponenseket, amelyekből felépíthetjük a CashUnit komponensünket.
- Tegyük fel, hogy találtunk négy olyan komponenst a piacon, amelyekből a 2.22. ábrán látható módon felépíthetjük a CashUnit komponensünket.
- A 2.22. ábra az árusító automata azon részét mutatja csak, amelyet a tervezési finomítás érintett.
- A CasUnit komponens kifejlesztésére **ugyanazokat a fejlesztési lépéseket végre kell hajtanunk**, amelyeket a VendingMachine komonensre végrehajtottunk.

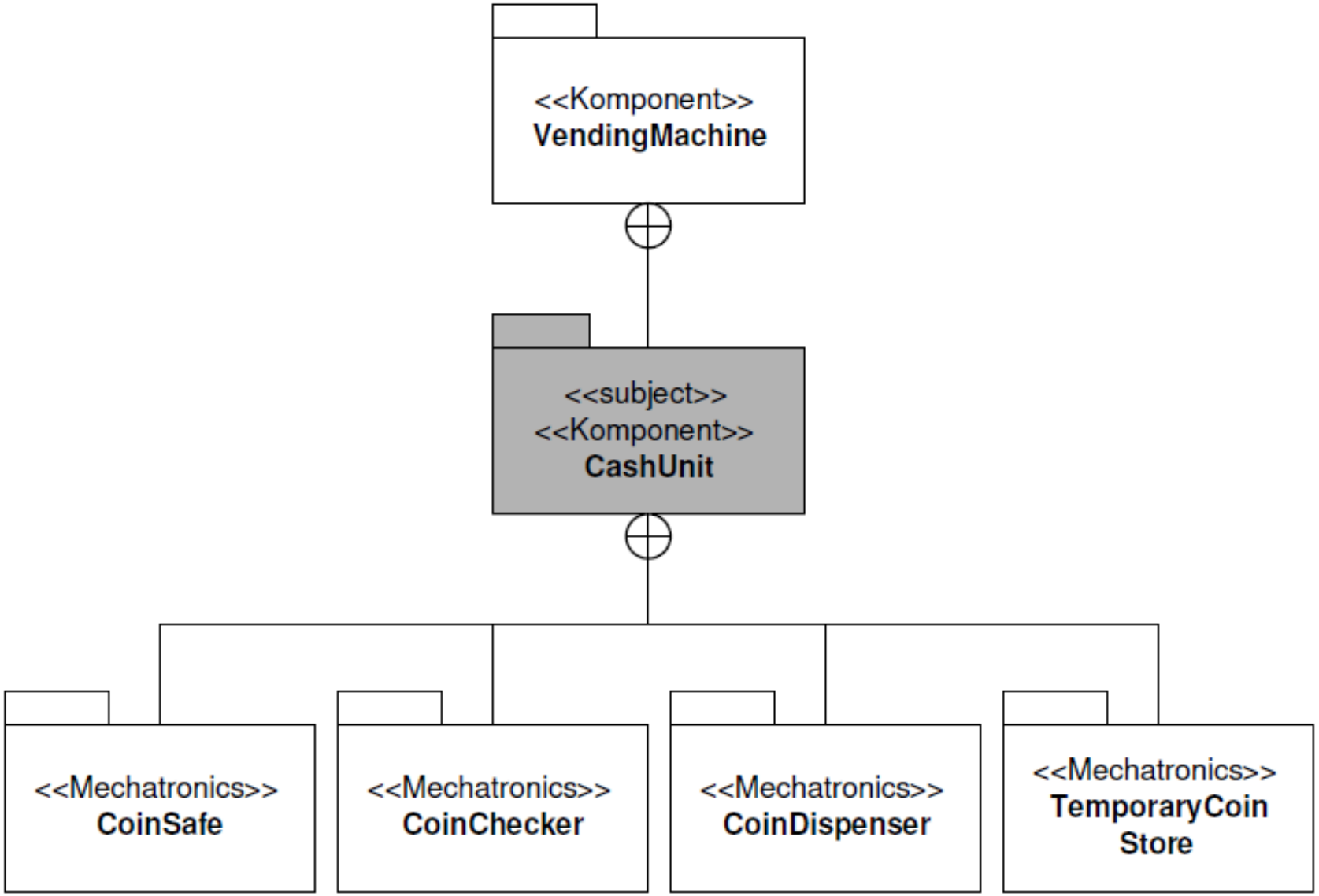

Fig. 2.22. Containment hierarchy for the *CashUnit*

COTS komponens integráció

- Egy létező komponenst akkor használhatunk fel újra legegyszerűbben, ha a specifikációik ekvivalensek azaz az igényelt komponens és a felhasználásra kiválasztott komponensek esetében a specifikációk ekvivalensek.
- A 2.23. ábra baloldala azt az általános esetet mutatja be, amikor a felhasználó a fejlesztést ezen szempontok figyelembevételével készíti el.
- A 2.23. ábra jobboldala ellenben azt az esetet mutatja be, amikor szintaktikus és szemantikus eltérés van az elvárt és a szolgáltatott interfészek között.

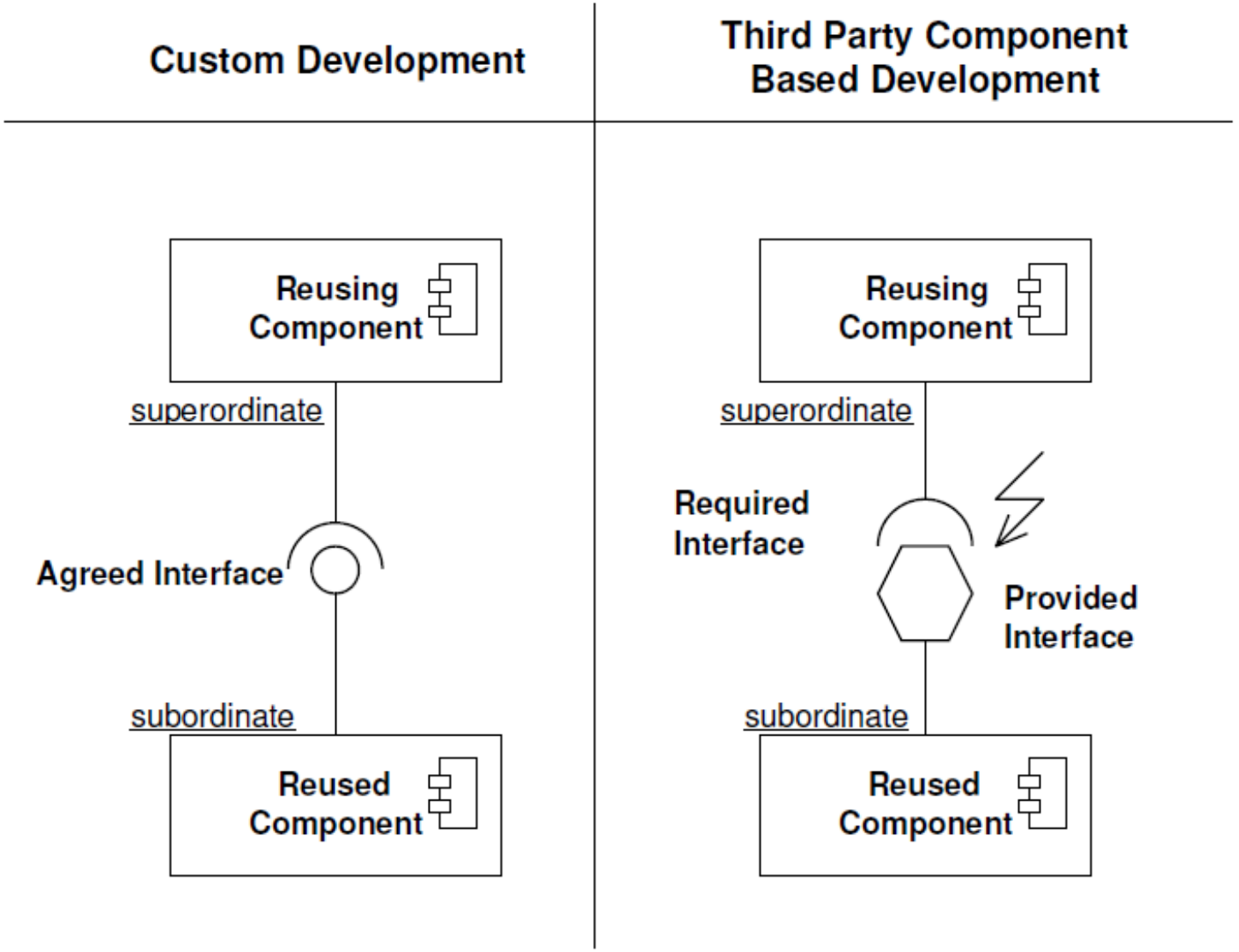

Fig. 2.23. Custom-designed component integration vs. third-party component integration

Megfelelési térkép

- A komponensek integrációját segíti elő az úgynevezett megfelelési térkép (conformance map), amely
  - Leírja egy COTS komponens kívülről látható tulajdonságait egy olyan leképezés segítségével, amely megadja azt, hogy a COTS komponens kifejlesztésekor használt jelölés és az általunk használt között mi a kapcsolat.
  - Ezt a leképezést csak akkor lehet végrehajtani, ha az újra felhasználni szánt komponens dokumentációjából kinyerhető információ korrekt és teljes a struktúra, a viselkedés és a funkcionalitás tekintetében.
    - Ezek az információk általában rendelkezésre állnak egy komponens esetében, de az információ elosztott lehet a rendszerben és ezért gyakran nehéz kinyerni és ezáltal a leképezést meghatározni.
- Csak akkor tudjuk eldönteni, hogy egy komponens megfelel-e céljainknak, ha a megfelelési térképet össze tudjuk állítani és az interfészek összehasonlíthatóak.

Szemantikus térkép

- Ha egy megfelelési térkép összeállítása után pozitív a döntésünk egy komponens újra felhasználása tekintetében, akkor a következő lépés a szemantikus térkép létrehozása.
- A szemantikus térkép a két komponens (a megtervezett és a felhasználni szánt) specifikációjának hasonlóságaira és különbözőségeire koncentrál és megpróbálja modellezni a leképezést a két eltérő interfész között.
  - A gyakorlatban ez egy wrapper (burkoló) komponens specifikációját jelenti, amellyel az eredeti COTS komponenst beburkoljuk.
  - Általánosabban ezt egy komponens átalakítónak (adapter) hívják és a KobrA fejlesztési elveinek megfelelően definiálható.

Az integrációs folyamat

- A harmadik parti (third-party) komponens integrálási folyamatát átalakító segítségével a 2.24. ábra mutatja be.

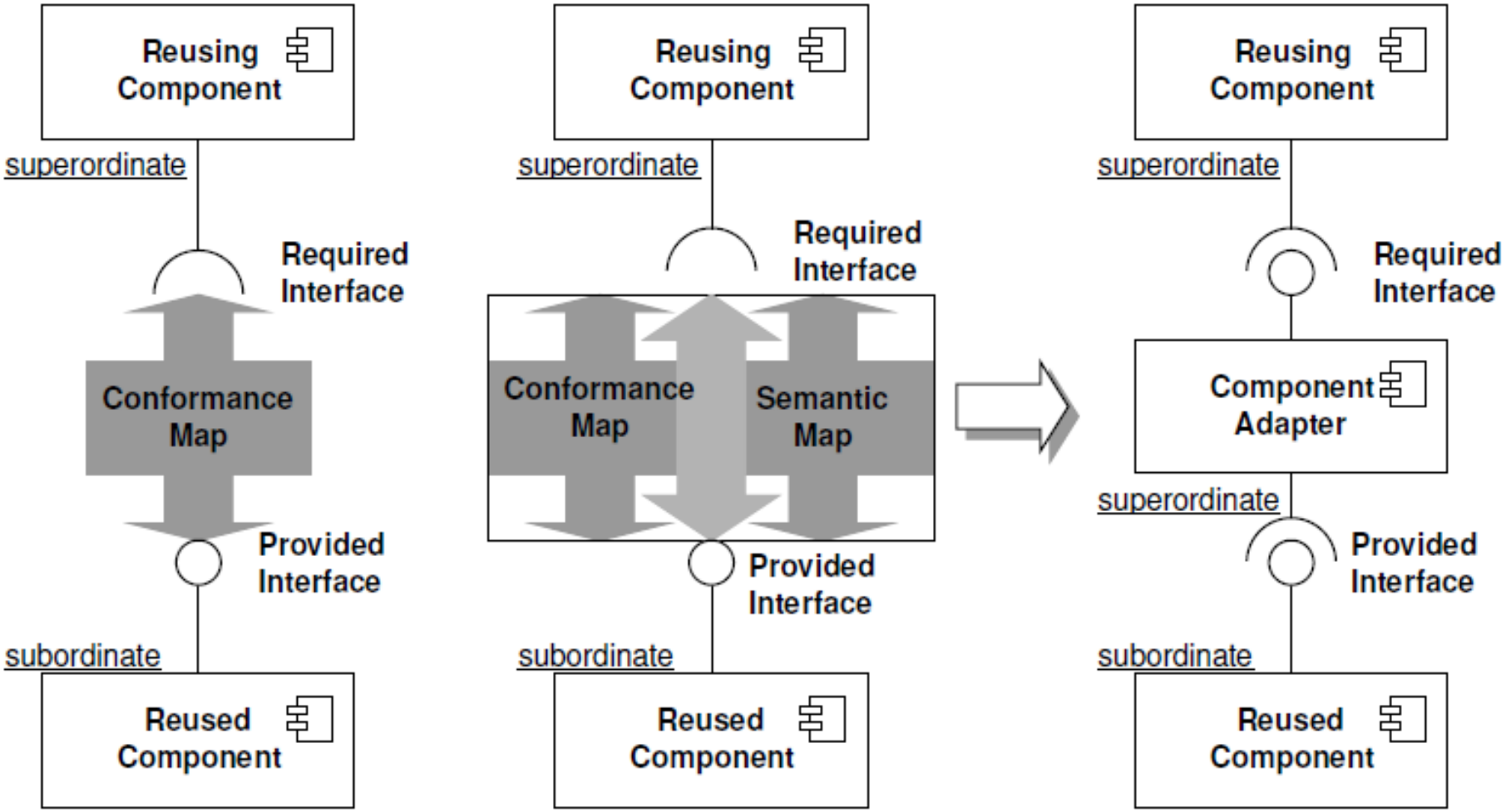

Fig. 2.24. Third-party component integration with adapter

A fizikai rendszer felépítése és telepítése

- A megtestesítés folyamatának két fontos lépése van még hátra:
  - A fizikai komponensek megkonstruálása és
  - telepítésük az aktuális célplatformra.
    - Különböző telepítési forgatókönyvek léteznek a telepítésre.
      - Egy vagy több logikai komponenst egy fizikai komponenssé transzformálunk
      - Egy logikai komponenst több fizikai komponenssé transzformálunk.
      - Több logikai komponenst több fizikai komponensé transzformálunk.
  - Nincsenek pontos irányelvek arra, hogyan konstruáljuk meg és telepítsük a fizikai rendszerünket.

- Termékcsalád koncepció

  - Product Family Concepts

Termékcsalád koncepció

- Eddig a komponensek kifejlesztését abból a szempontból vizsgáltuk meg, hogy egy adott probléma megoldásához fejlesztettünk ki komponenseket és azt a kérdést vizsgáltuk meg, hogy milyen módon használhatók fel újra.
- Most azzal a kérdéssel foglalkozunk röviden, hogy milyen elvek szerint lehet eleve újrahasználható generikus komponenseket létrehozni.
- A termékcsalád koncepció ezt az elvet teszi meg központi kérdésnek.
  - Ebben az esetben az újrafelhasználás kérdése az architektúra szintjén jelenik meg.
  - Egy termékcsalád egy generikus rendszer vagy pontosabban egy generikus komponens keretrendszer, amely alkalmas több hasonló rendszer létrehozására.

Termékcsalád mérnökség

- A termékcsalád mérnökség nem más mint egy felhasználó által tervezett komponens újrafelhasználásának egy szervezési módja.
- A 2.25. ábra ezt a folyamatot illusztrálja.

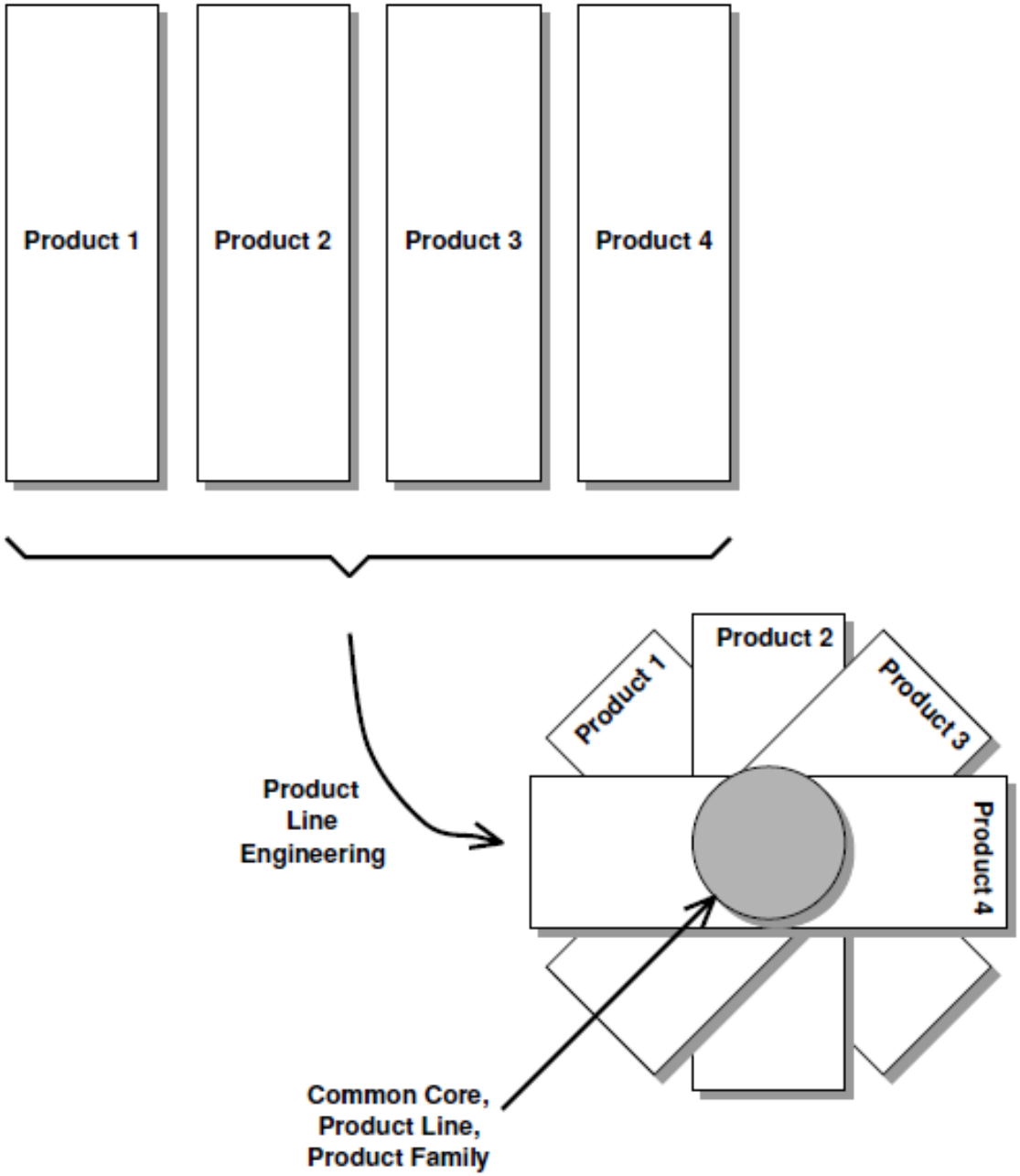

Fig. 2.25. Product line as intersection of different but similar individual products

Döntési modellek

- Azt eldönteni, hogy miből lehet termékcsaládot készíteni nem könnyű feladat. Ehhez számba kell venni az egyedi termékek közös tulajdonságait is és a különbözőségeiket is, ez azonban még kevés. Azt is tudnunk kell, hogy a változó tulajdonságok közül melyek tartoznak egy konkrét termékhez. A **döntési modelleknek** itt van jelentősége.

A döntési modellek szerepe

- Egy döntési modell minden döntéshez három dolgot definiál
  - Egy szöveges kérdést, amely arra a területre vonatkozik, amellyel kapcsolatban a döntést meg kell hozni;
  - A lehetséges válaszok egy halmazát, amelyek mindegyike a termékcsalád egy-egy specifikus példányára képeződik le;
  - Azt a helyet ahol a döntés megtestesül.
- A 2.6. táblázat egy döntési modelljét írja le a 2.7. ábrán bemutatott, az árusító automata környezeti térképéhez tartozó strukturális modellnek.

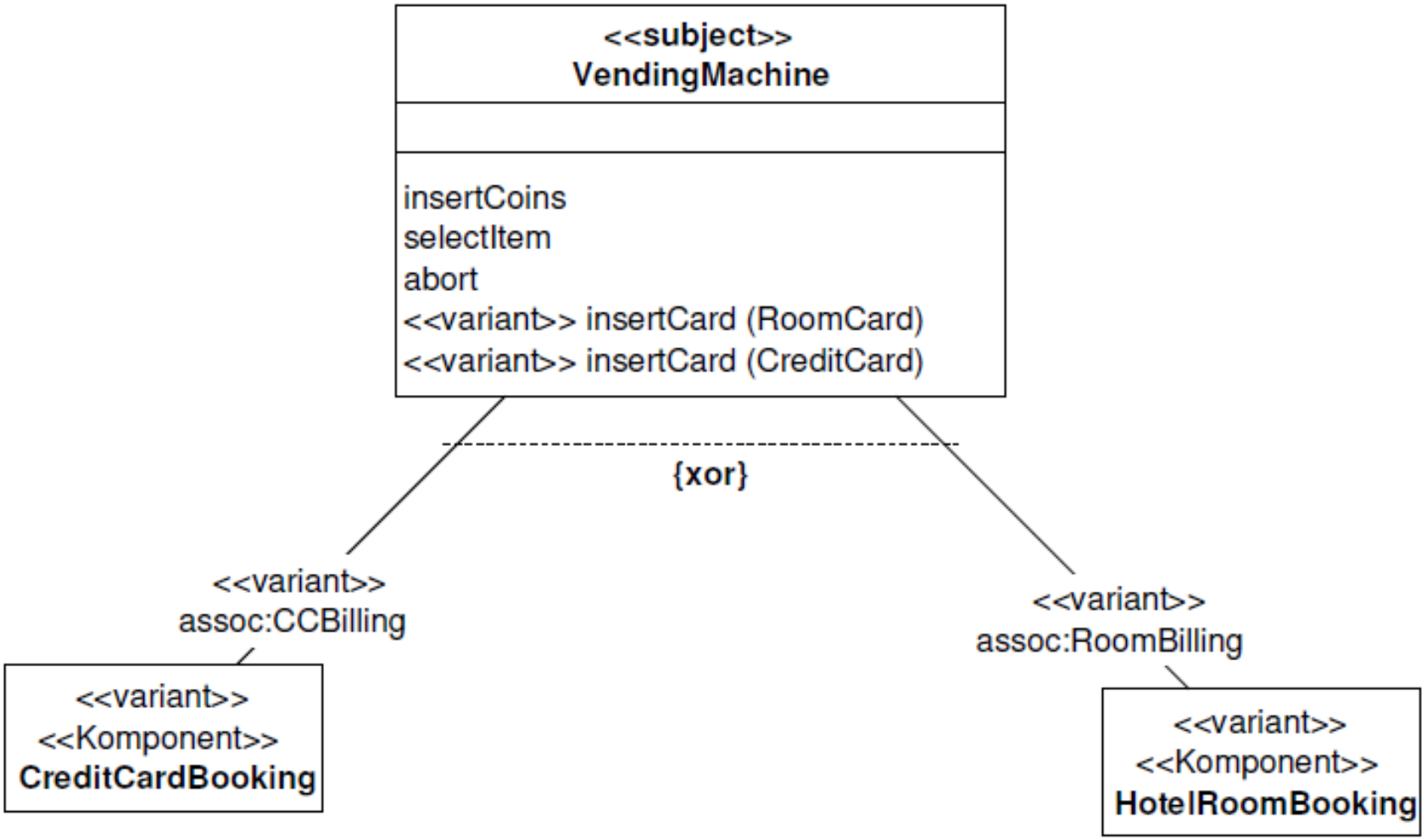

Fig. 2.7. Context realization structural model for the vending machine context

Table 2.6. Decision model according to the context realization structural model of the *VendingMachine*

No.	Question	Variation Point	Resolution	Effect
1.a	CCBilling supported?	VendingMachine	no (default)	remove Component CCBilling
			yes	remove stereotype ≪variant≫
1.b	RoomBilling supported?	VendingMachine	no (default)	remove Component RoomBilling
			yes	remove stereotype ≪variant≫
	1.a and 1.b are alternatives according to the ≪xor≫ stereotype			

A termékcsalád és a döntési modellek kapcsolata

- Egy adott termékcsaládhoz tartozó minden egyes végtermék előállítását döntési modellek támogatják.
- A szoftverfejlesztési folyamat során létrejött modellekhez döntési modelleket kell készíteni:
  - a környezeti térkép;
  - a komponens specifikáció;
  - a komponens realizáció összes modellre.
- Az alkalmazásért felelős mérnöknek a döntési modell kérdéseire pontos választ kell adnia.
  - Más-más válasz esetén más és más termék jön létre.
- A termékcsalád koncepció egy új fejlesztési dimenziót hoz be, amelyet a 2.26. ábrán láthatunk.

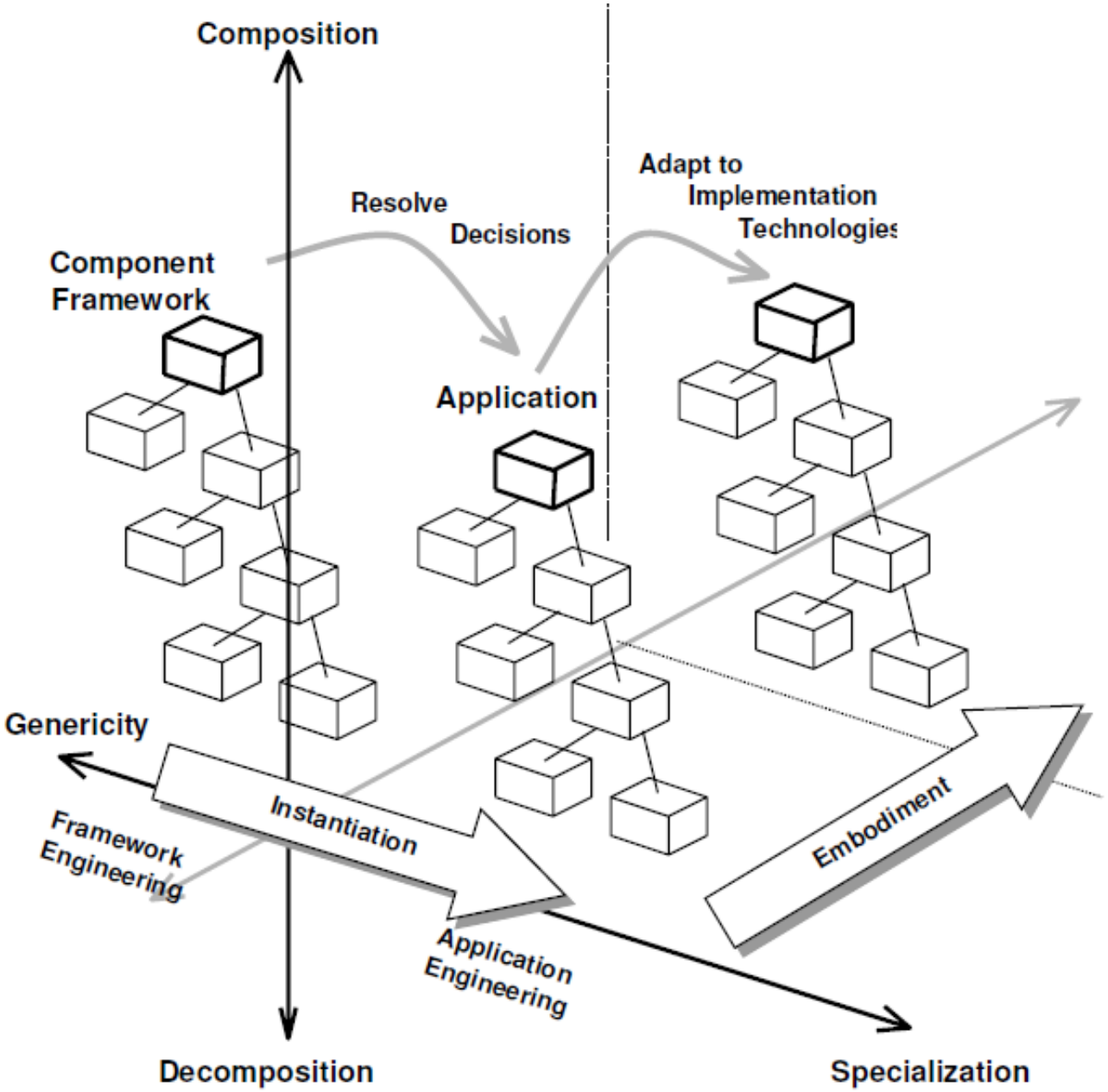

Fig. 2.26. Instantiation of a final application out of a component framework of a product line

Az új dimenzió

- A 2.26. ábra világosan szeparálja az alkalmazás és a keretrendszer mérnökségeket.
- Az alkalmazás mérnökség a közös mag példányosításával foglalkozik.
- A keretrendszer pedig a közös mag kifejlesztésével.

Keretrendszer mérnökség

- Egy termékcsalád közös magjának kifejlesztésére koncentrál.
- Egy keretrendszer kifejlesztésénél ugyanazokat a fejlesztési, modellezési és tervezési lépéseket kell végrehajtani, mint egy egyedi rendszer létrehozásakor.
- Egy keretrendszer nem más mint komponensek olyan nem teljes összeállítása, amelyet további komponensekkel teljessé kell tenni, hogy egy egyedi rendszer előálljon.
- Egy egyedi rendszer tehát egy generikus termékcsalád egy példánya.

Generikus komponens egy variánsa

- Bármely olyan sajátosságot, amely egy egyedi végső termék jellemzője egy generikus komponensen belül, <<variant>> sztereotípiával jelölünk.
- A változó (variant) tulajdonságok tesznek egyedivé egy terméket a termékcsaládon belül.
- A döntési modell támogatja a variálhatóságot egy termékcsaládon belül.
- A 2.25. ábrán az árnyékolt részek jelzik a közös részeket a termékcsaládon belül.
  - Ezek a részek ugyanúgy modellezhetők, ahogy azt az eddigiekben vázoltuk.
  - Az ábrán nem árnyékolt részek felelnek meg a változó tulajdonságoknak.

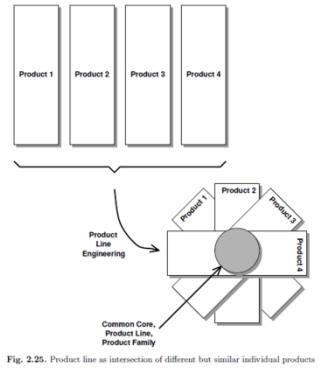

Fig. 2.25. Product line as intersection of different but similar individual products

Az árusító automata

- A 2.7.ábra az árusító automata két változatát írja le. A **CCBilling** és a **Roombilling** egyike sem tartozik a **VendingMachine** generikus modelljéhez.
- A változatok az absztrakciós szintek bármelyikén megjelenhet.
- Lehetnek változó
  - alrendszerek,
  - komponensek,
  - attribútumok,
  - műveletek.
- Bármely változatot a modell szintjén kell elsődlegesen definiálni.
  - Lehetnek változó használati esetek a használati modellben, változó komponensek és osztályok a strukturális modellben, valamint változó funkcionalitás a viselkedési és funkcionális modellekben.
- A 2.27. ábra a környezeti térképben a használati eset egy változatát mutatja be.

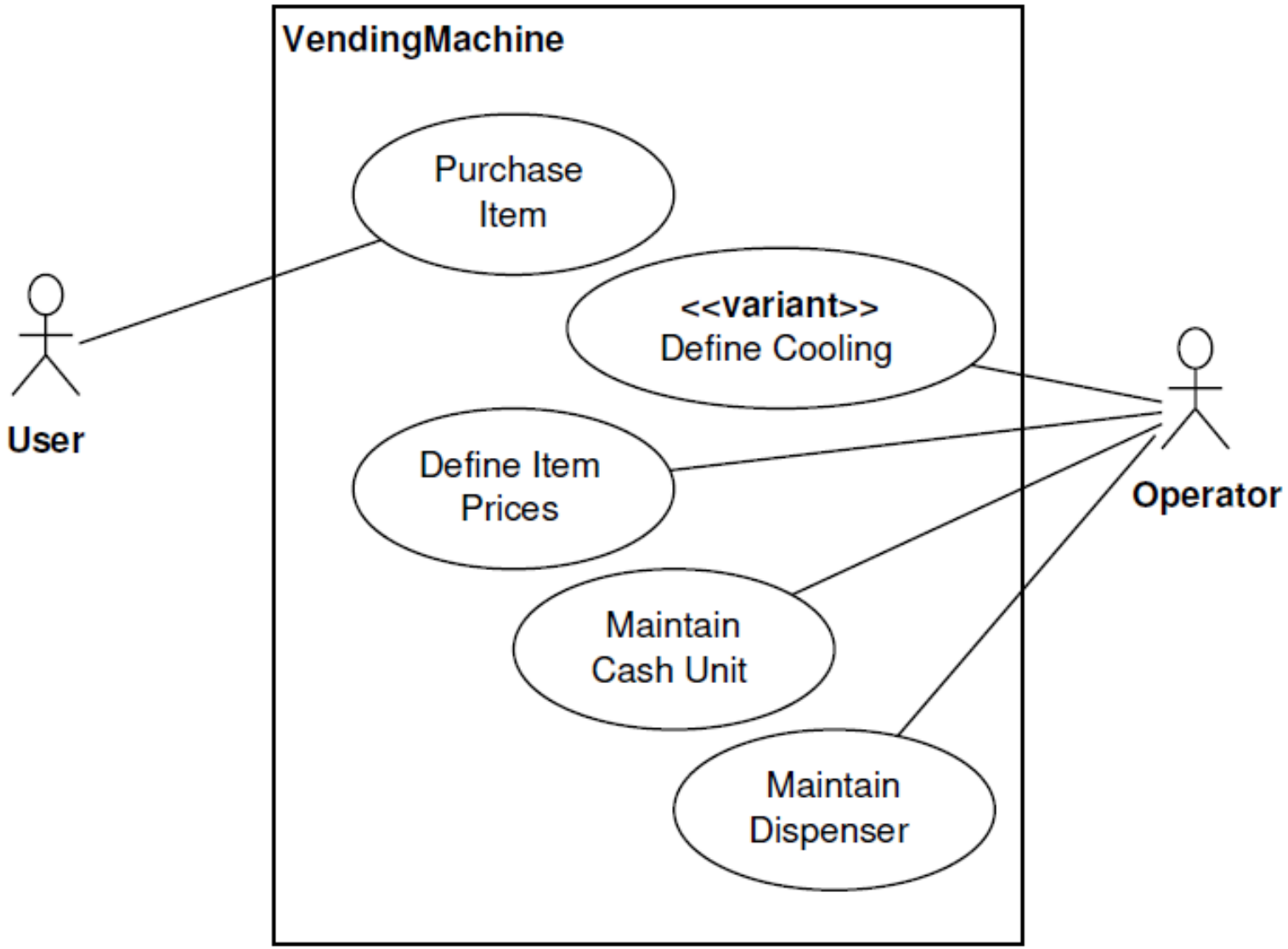

Fig. 2.27. Example of a usage model with a variable use case

Magyarázat a 2.27. ábrához

- A <<variant>>DefineCooling a végső termék számára specifikálja a műveletet úgy, hogy az lehetőséget ad a szerviz szakembereinek arra, hogy a hűtést az árusító automata lokális interfésze alapján állítsák be.
- Egy másik példányban ezt esetleg nem engedik meg, mert a hűtést egy előre definiált értékre állítják be.

Dokumentálás és minőségbiztosítási tervek

Minőségbiztosítási követelmények

- Pontosan definiálni kell, hogy a minőség fogalma mit jelent az adott fejlesztési projekt szempontjából és ez hogyan manifesztálódik a különböző fajta termékekben
- Pontosan le kell írni, hogy a termék mely minőségi aspektusai fontosak és milyen minőségi szinteket különböztetünk meg
- Egy tervet és eljárást kell készíteni a fenti elemek rendszerbe szervezésére.

Dokumentálás a komponensek szintjén

- 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.

Összefoglalás

- Minden szoftverfejlesztő projektnek egy helyes fejlesztő módszeren kell alapulnia.
- Ilyen módszert reprezentálhat egy fejlesztő keretrendszer.
- Egy ilyen keretrendszer a KobrA módszer, amelyet az előzőekben nagyvonalakban bemutattunk.
  - A KobrA módszer legfontosabb jellemzője az, hogy konkrét irányelveket ad arra nézve, hogy hogyan használjuk az UML-t egy komponens alapú szoftverfejlesztési projektben.