Elosztott Rendszerek és Szakterületi Modellezés jegyzet

Bevezetés

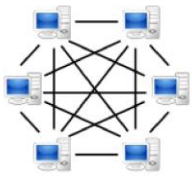
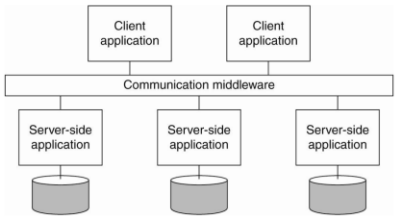
Elosztott rendszerek definiciója

- Leslie Lamport definiciója: “A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”
- Wolfgang Emmerich: „Elosztott rendszernek nevezzük az autonóm, egymással hálózaton együttműködő számítógépeket, amelyek közös együttműködésén alapuló szolgáltatását a felhasználó úgy érzékeli, mintha azt egy integrált eszköz szolgáltatta volna.”

Elosztott vs központosított rendszerek

- Elosztott
  - Rugalmasság
  - Skálázhatóság
  - Helyi önállóság
  - Jobb megbízhatóság és elérhetőség
  - Nagyobb teljesítmény
  - Biztonsági hiányosságok lokalizálhatósága
- Központosított
  - Egyszerűbb és biztonságosabb adminisztráció
  - Megbízhatóság és elérhetőség
  - Képzett gárda

Elosztott formák

- Cluster
- Grid
- Transaction Processing Systems
- P2P
  - 
- Enterprise Application Integration
  - 

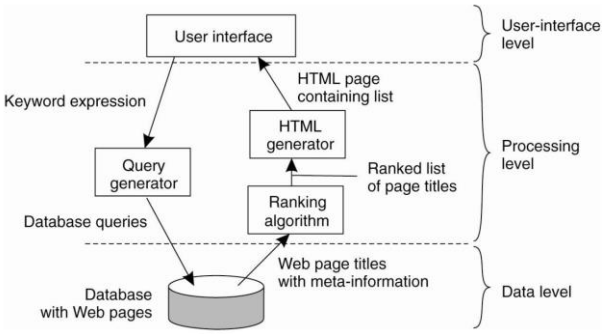
- SOA (Service Oriented)
- Sensor Networks

Átlátszó elosztottság

- Teljes átlátszóság: Az elosztottsággal behozott komplexitás teljes mértékű lefedése. Általában operációs rendszer szinten kell megvalósítani.
- Részleges vagy szelektív átlátszóság: A teljes átlátszóság alkalmazása nem teszi lehetővé, hogy az alkalmazások kihasználják az egyes elemek elhelyezkedésének ismeretéből származó esetleges előnyöket.

- A teljes átlátszóság megvalósításának legnagyobb hátránya a komplexitás megnövekedése. Az ISO 10746-3 szabvány pontosan meghatározza, hogy milyen 10 általános funkciót kell megvalósítani ennek eléréséhez.
- A 10 átlátszó funkció
    - Átlátszó hozzáférés
    - Helyfüggetlenségi átlátszóság
    - Konkurens működés átlátszósága
    - Átrendezés (migráció) átlátszóság
    - Átlátszó replikáció
    - Átlátszó partíciónálás
    - Állandósági átlátszóság (persistence transparency)
    - Hibatűrési átlátszóság
    - Teljesítmény átlátszóság
    - Skálázási átlátszóság
- Átlátszóság
    - Hely-alapú: A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy hol van a végrehajtó szerver
    - Funkcionális: A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy milyen algoritmusok, illetve optimalizálási eljárások vannak a távoli szerveren.

Elosztott információs rendszer részei

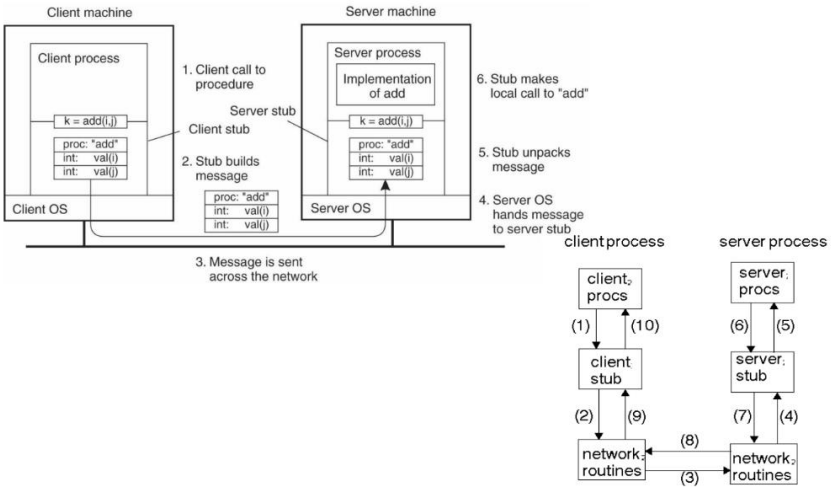


- 3 szint
    - Prezentáció – User Interface Level
    - Feldolgozás – Processing Level
    - Adat – Data Level
- Működési feltételek
    - Elkülönítési mechanizmus
    - Objektum azonosítás
    - Biztonság

RPC (Remote Procedure Call)

- Átláűtszoság
    - Hely-alapú: A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy hol van a végrehajtó szerver
    - Funkcionális: A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy milyen algoritmusok, illetve optimalizálási eljárások vannak a távoli szerveren.

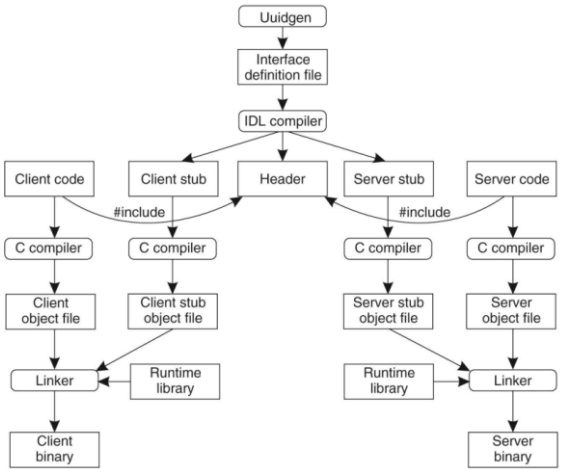
RPC hívások 10 lépése



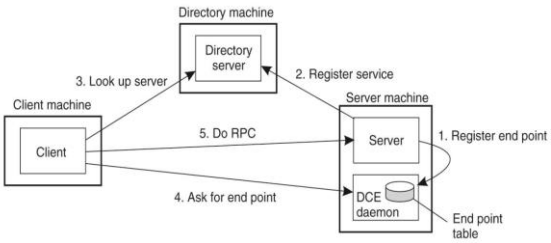
- Runtime

1. Client call to procedure..
2. Stub builds message
3. Message is sent across the network.
4. Server OS hands message to server stub..
5. Stub unpacks message.
6. Stub makes local call to "add"
7. Add executes and returns answer to server stub.
8. Server stub composes reply message containing answer.
9. Message is sent across the network.
10. Client OS passes reply to client stub.
11. Client stub unpacks and returns result.

RPC fejlesztés



- Problémák
  - Request from cli -> srv lost
  - Reply from srv -> cli lost
  - Server crashes after receiving request
  - Client crashes after sending request
- Kötés



COM (Component Object Model)

COM jellemzők

- Dinamikus linkelés
- Szeperáljuk az interfészt és implementációt külön osztályokba
- Az interfészben nem szerepeltethetünk változókat
- Az interfész metódusai „pure virtual”
- Egy interfész egyszerre csak egy interfészből származhat, de akár hányszor
- Az interfész kiterjeszthető
- Egy implementáció több interfészt implementálhat
- A komponens működéséhez szükséges műveletek egy generikus interfészbe kerülnek, ebből származik minden interfész (root interfész)
- A COM root interfésze IUnknown (QueryInterface, Addref, Release)
- Az interfész metódusok visszatérési értéke HRESULT
- Minden Interfészbeli metódusokban, minden paraméternek van attribútuma
- A COM hívásai alapvetően szinkron típusúak
- COM speciális interfészeket kínál (IClassFactory, IConnectionPoint, Idispatch)

COM hiányosságok

- Csak Windows
- Alapvetően szinkron
- Apartman kiegészítés, bonyolult
- Nincs tranzakciókezelés
- Rövid élettartamú objektumok létrehozása időigényes
- Túl kevés vagy túl sok szál – nem hatékony
- Nem elég kifinomult a biztonsági modell
- Lazán csatolt rendszerekhez túl szigorú
- Nem elég rugalmas eseménykezelés

Middleware Szolgáltatások

- Olyan szolgáltatások, melyeket általában a középső réteg valósít meg.
- Ezek nélkül nem létezik alkalmazás szerver!
- Middleware szolgáltatások
    - Távoli eljáráshívás
    - Szálkezelés
    - Terheléskiegyenlítés
    - Átlátszó hibakezelés
    - Perzisztencia
    - Tranzakciókezelés
    - Objektumok életciklusa
    - Aszinkron üzenetkezelés
    - Biztonság

- Middleware típusok
    - Explicit
        - Alapvetően API hívások révén
        - Tipikusan a CORBA használja
    - Implicit vagy deklaratív
        - Attribútumok(COM+)
        - XML Descriptor (EJB)
    - Természetesen az implicit middleware a preferált típus. Szétválik a fejlesztés és az adminisztráció (Component explorer)

COM+ Middleware szolgáltatások

- Threading
- Applications Running as Service Applications
- Events
- Context
- Just-in-Time Activation, Object Pooling
- Object Constructor Strings
- Queued Components
- Security
- Synchronization, Context
- Transactions

.NET

Remoting

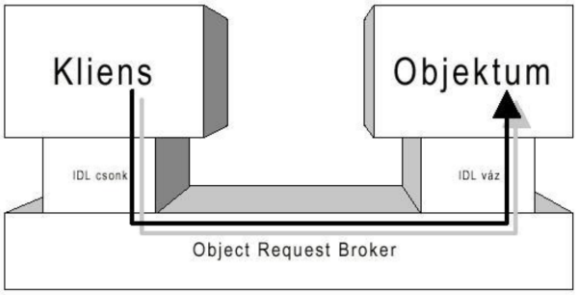
1. AppDomain: alkalmazástartományok közötti kommunikációra
    - Alapból minden alkalmazásban létrejön egy default alkalmazástartomány, amiben fut a kódunk. De egy processzben lehet több is. Egymástól elszigetelten futnak. Nem férnek hozzá egymáshoz.
    - Védetten futnak egymástól
    - Lehet szabályozni az egyes alkalmazástartományok jogosultságait
    - Hatékonyabb, mintha külön processzekben futnának (gyorsabban lehet váltani közöttük)
    - Egy alkalmazástartományt el lehet távolítani az alkalmazásból, ha nem kell. Egy betöltött assembly-t nem.
2. Architektúra
    - Szereplők
        - Remotable object: A távolról elérhető objektum.
        - MarshalByRefObject ősosztályból kell származnia.
        - Szerver: Hosztolja a remotable object-et.
        - Kliens: a szerveren hosztolt távoli objektumhoz kapcsolódva meg tudja hívni annak metódusait. A kliens a távoli objektumot egy helyi proxy-n keresztül látja.
        - Fontos: a távoli objektum megosztása a kliens és a szerver között:
        - A kliens és a szerver számára is rendelkezésre kell állnia a
        - távolról elérhető objektum implementációjának.
        - Verziózási okokból sokkal előnyösebb, hogyha a kliens és a szerver csak egy interfészt osztanak meg egymás között, amit a távoli objektum implementál. Ezt egy kicsit bonyolultabb használni
    - Kommunikáció kliens és szerver között
        - Ahhoz, hogy a kliens és a szerver kommunikálni tudjon egymással, három dologban egyet kell érteniük, ezt el kell dönteni minden esetben:

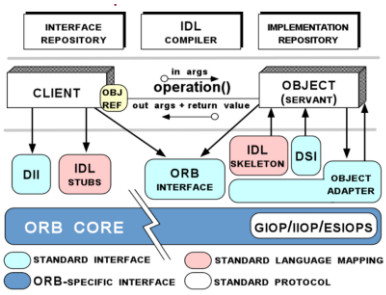
        - Channel: definiálja a kommunikációs protokollt. Ezen a csatornán keresztül zajlik a kommunikáció. Alkalmazás (tartomány) szinten kell regisztrálni a csatornákat. Legalább egy csatornát be kell regisztrálnia a szervernek. Ugyanarra a portra két csatorna nem regisztrálható.
        - Activation mode: a távoli objektum élettartamának kezelésének a módja. Mikor jöjjön létre szerver oldali objektum? Melyik kérést melyik objektum szolgálja ki?
        - Formatter: azt határozza meg, hogy milyen formában sorosítódjon az adat a hálózati adatfolyamba.
        - (Cím, url)
    - Csatornák
        - IPC, TCP, HTTP
    - Aktiválási módok
        - Client Actiovation (CAO – Client Activated Object, Activated): A kliens kérésére jön létre az objektum a szerver oldalon.
        - Server Activation (SAO – Server Activated Object, Well known): A szerver kontrolálja az objektumok létrehozását. Két esete van:
            - SingleCall: minden kliens kéréshez létrejön egy új objektum szerver oldalon. A metódus meghívása után nincsen már rá szükség, a GC eltávolítja.
            - Singleton: a szerveren egy példány jön létre az objektumból, és ez szolgál ki minden klienst.
    - Formatters: Formázók
        - Binary, SOAP
    - Távoli objektumok címzése
        - Szerver oldali csatornának megfelelő protokoll (http, tcp, ipc)
        - A szerver címe
        - A csatornához tartozó portszám
        - A remoting szerver alkalmazásban definiált alkalmazásnév (egy gépen több remoting alkalmazás is futhat egyszerre)
        - SAO esetben a szerver alkalmazáson belül minden távoli objektum rendelkezik egy további, egyedi objectUri-jal (ezért hívjuk Wellknown-nak).
        - CAO esetben elég csak a szerver alkalmazást megcímezni, a szerver majd létrehozza az objektumot a kliens kérésé
3. Működés
4. Kiterjeszthetőség

WCF

CORBA

CORBA alap architektúra





- CORBA részei
  - Application (Kliens)
  - Facilities
  - Services: Middleware – explicit és Implicit
  - ORB (Object Request Broker)

COM – CORBA összehasonlítás

- COM csak Windows-on, CORBA platform független
- COM-ban nincs tranzakció kezelés, CORBA-ban van
- CORBA-ban kifinomultabb biztonsági modell, mint COM-ban
- COM implicit middleware, CORBA explicit middleware
- COM root interfész: IUNKNOWN, CORBA root: CORBAObject
- COM-ban egyszeres öröklés, másikban többszörös
- COM-ban az attribútumok nem támogatottak, CORBA-ban get;set;-re fordul

CORBA Services

- Lifecycle Service
- Relationship Service
- Naming Service
- Persistent Object Service
- Externalization Service
- Event Service
- Object Query Service
- Object Properties Service
- Object Transaction Service
- Concurrency Service
- Licensing Service
- Trader Service
- Security Service
- Time Service
- Collection Service
- Notification Service

ORB

- A CORBA runtime
- Szolgáltatásokat nyújt a kliens-objektum kommunikációhoz
- ORB API-t kínál
- Statikus és dinamikus interfészt biztosít
- Kezeli az interfész és Implementációs Repository-t
- Az objektum adapterekkel kommunikál
- Lokális és távoli hívások kezelése

Az Objektum adapterek

- Implementációk regisztrálása,
- Objektumreferenciák létrehozása és generálása,
- Objektumreferenciák leképzése az hozzájuk tartozó implementációkra,
- Implementációk aktiválása és deaktiválása,
- Kérések továbbítása a vázon vagy a DSI-on keresztül,
- A biztonság szavatolása a Security Service segítségével.
- Két alap típusa BOA, POA
    - BOA: Basic Object Adapter
        - Négy féle kapcsolatot tesz lehetővé a szerver és az objektumok között:
        - Osztott (Shared): olyan szerver, amely több objektumot tartalmaz.
        - Állandó (Persistent): ugyanolyan, mint a Shared, csak aszerver a BOÁ-tól függetlenül aktiválandó, például installáláskor.
        - Egyedi (Unshared): szerverenként legfeljebb egy objektum lehet aktív.
        - Funkciónkénti szerver (Server-per-method): a BOA minden funkció felhívás alkalmával elindít egy új szervert, amely a funkció befejeztével leáll.
    - POA: Portable Object Adapter
        - Összetettebb és fejlettebb mint a BOA, a POA egy pontosítást jelent a BOA-hoz képest.
        - Általános objektum menedzsment célokra lett kifejlesztve
        - Hiearchikus, több POA lehet egyszerre a rendszerben egy jól definiált hierarchia segítségével
        - Root POA

GIOP Protokoll

- General Inter ORB (Object Request Brokers) Protokoll
- Egyszerűség
- Méretezhetőség
- Alacsony költség
- Általánosság
- Architekturális függetlenség

GIOP részei

- Az általános adatábrázolási definíció (Common Data Representation - CDR)
    - Byte sorrendiség
    - Adatillesztés
- A GIOP üzenetek formátuma
    - 7 üzenet, 3-kliens-szerver, 3 szerver kliens és 1 kétirányú
- A GIOP szállítási feltételek
    - Kapcsolat alapú protokoll
    - Megbízható adattovábbítás
    - A kapcsolat felépítése hasonlóan kell lezajlania, mint a TCP/IP protokollnál.

GIOP üzenetek

- RequestMessage: ez a kérést továbbító üzenet, három részre tagolódik, a GIOP fejlécre, a kérés fejlécre, és a kéréstörzsre. A kérés fejlécében található a szolgáltatás kontextus (a Security és Transaction Service számára tartalmaz információkat), a kérés azonosítója, az objektumazonosító, a felhívandó funkció neve és esetleg más információk. A kérés törzse a funkció összes in és inout irányú paraméter értékét foglalja magában.
    - K -S
- ReplyMessage: ez a befejezett kérésre való reakciót továbbítja a kliens felé. Mint minden GIOP üzenet, ez is GIOP fejléccel kezdõdik majd ugyanúgy, mint a kérés üzenet további két komponens következik. A Reply

Header szintén tartalmaz szolgáltatás kontextust, kérésazonosítót, mely azonos a megfelelõ RequestMessage kérésazonosítójával. Végül a Reply Header-ben az utolsó adat a státusz kód, ez határozza meg, hogy a ReplyMessage harmadik komponense, a Reply Body milyen információkat tárol. Amennyiben a státuszkód no_exception értéket tartalmaz, akkor a Reply Body a felhívott funkció összes out és inout paraméterét tartalmazza. Ha a státuszkód System_Exception vagy User_Exception értékû, akkor a válasz törzse a kivétel információit tárolja. A státuszkód felveheti a Location_Forward értéket is, ilyenkor a válasz törzséből az ORB kiolvashatja a megcímzett objektum új címét és automatikusan az új címre továbbítja a kérést.
    - S - K
- CancelRequest: ez az üzenet utasítja a szervert egy feldolgozás alatt álló RequestMessage vagy LocateRequest üzenet végrehajtásának törlésére. Szerkezete a szokásos GIOP fejlécbõl és egy kérésazonosítóból épül fel. A kérésazonosítóból tudja meg a szerver, hogy melyik üzenet feldolgozását fejezze be.
    - K - S
- LocateRequest: ennek az üzenetnek az a szerepe, hogy egy kliens megtudja egy szervertõl, hogy az adott objektumreferencia még érvényes-e, illetve, hogy a szerver képes-e feldolgozni egy adott kérést. Ez az üzenet tulajdonképpen nagyon hasonló a RequestMessage üzenethez, azzal az egy kivétellel, hogy nem tartalmazza magát a kérést, így elkerülhetõ a kérés elküldésekor nagyobb méretû adatforgalom.
    - K -S
- LocateReply: a Locate Request üzenetre a szerver ezzel az üzenettel válaszol. Az üzenet felépítése nagyon hasonló a RequestReply-hoz, azzal a kivétellel, hogy nem tartalmazza a visszaadott paraméterek értékét, mivel ez az üzenet nem válasz valamilyen kérésre. Amennyiben a státuszkód Location_Forward, akkor természetesen az üzenet törzsében találhatunk egy IOR-t, mely az objektum új címét tárolja.
    - S - K
- CloseConnection: ennek az üzenetnek a segítségével jelzi egy szerver, hogy befejezi kapcsolatát az ORB-bal. Ezt természetesen csak akkor teheti meg, ha nincs folyamatban kérés, természetesen érkezhetnek kérések a CloseConnection üzenet elküldésével párhuzamosan, ilyenkor az ORB a klienseknek a Completed_Maybe (“talán befejezett”) kivétellel jelzi, a kérés teljesítésének a státuszát. Ez az üzenet csak a GIOP fejlécbõl áll, hiszen további információkra nincs igazán szükség.
    - S - K
- MessageError: ezt az üzenetet mindkét fél küldheti, ezzel jelezve bármilyen GIOP üzenet hibás voltát. Ez az üzenet is csak egy GIOP fejlécbõl áll, azt feltételezve, hogy az üzenet küldõje tudja mit csinált rosszul.
    - SK – KS

CORBA hátrányok

- Eltérő implementációk
- Inkompatibilítás az implementációk között
- Explicit Middleware
- Laza szabványosítás
- Migrálhatóság nehézsége
- Skálázhatósági problémák
- Nehézkes
- Nincs jól definiált konfigurálási iránymutatás, a teljes rendszerek építésére

Lehetséges megoldás

- Component Middleware, CORBA Component Model (CCM)
- Container mechanizmus
- Szabványos konfigurálási és telepítési mechanizmusok
- Speciális interfészek támogatása
- Kommunikációs pontok

- Esemény „sinks” és források

Komponens és service különbség

- Szinkron – aszinkron

JEE (Java Enterprise Edition)

JEE Middleware szolgáltatások

- Többszálúság
- Tranzakciókezelés
- Biztonság
- Perzisztencia
- Névszolgáltatás
- Objektumok életciklusának kezelése
- Távoli metódushívás
- Aszinkron üzenetkezelés
- Skálázhatóság
- Terhelés-kiegyenlítés

JEE API-k

- Java Persistence API (JPA): objektum-relációs leképezés
- Enterprise JavaBeans (EJB): elosztott, üzleti logikai komponensek
- Java Message Service (JMS): aszinkron üzenetkezelés
- Java Transaction API (JTA): tranzakciókezelés
- Contexts and Dependency Injection for Java (CDI): függőséginjektálás
- Java Authentication and Authorization Service (JAAS): biztonság
- Webes technológiák:
    - Java Servlet
    - JSP (JavaServer Pages)
    - JSF (JavaServer Faces)
    - Webszolgáltatások
        - Java API for XML-Based Web Services (JAX-WS): SOAP alapú XML webszolgáltatások
        - Java API for RESTful Web Services (JAX-RS): REST stílusú webszolgáltatások

Elosztott adatfeldolgozás

Elosztott adatbázis

Több, logikailag kölcsönösen összefüggő adatbázisok összessége. Ezek az adatbázisok fizikailag különböző helyeken vannak és hálózaton keresztül kötődnek össze. Az elosztottság nem látható a felhasználó számára (transzparencia).

Elosztott adatfeldolgozás előnyei

- Összteljesítmény javulás
- Rendelkezésre állás növekedés
- Megbízhatóság javulás
- Skálázhatóság
- Rugalmasság
- Konfigurálhatóság
- Elosztott tranzakciók kezelése
- Átlátszóság

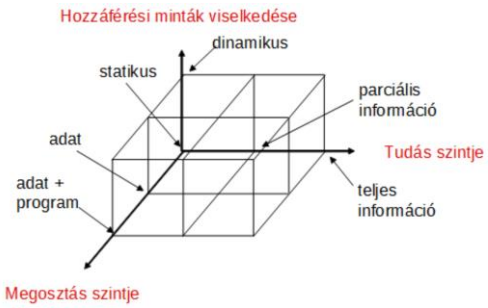
Autonómia

- Az autonómia nem az adat, hanem a vezérlés elosztottságára jellemző. A teljesen autonóm (önálló) rendszerrel szemben a következő követelmények támaszthatók:
    - Az adatbázis helyi műveleteit nem befolyásolja, a multiadatbázis rendszerben való részvétel.
    - Az adatbázis műveleteinek helyi feldolgozására és optimalizálására nincsenek hatással a több adatbázisra kiterjedő lekérdezések.
    - A rendszer globális konzisztenciáját nem befolyásolja, ha egy adatbázis csatlakozik a rendszerhez, vagy távozik belőle.
- A teljesen autonóm rendszerek egy szélsőséges esete a teljes elszigeteltség, amikor az egyes adatbázis kezelők nem tudnak a többi adatbázis létezéséről, és nem is kommunikálnak velük. Az ilyen rendszerek összekapcsolása meglehetősen nehéz, mivel nem adnak lehetőséget semmilyen szintű központi felügyeletre.
- Az önállóság másik végletét képezik a szorosan integrált rendszermegvalósítások, ahol úgy alakítják ki az egyes komponenseket, hogy az elosztottság ellenére azok kifelé egy egységes, központosított adatbáziskezelő képét mutassák.
- A két megoldás között helyezkednek el azok a fél-autonóm, lazán integrált rendszerek, melyek leginkább úgy jellemezhetők, hogy elsősorban önálló működést valósítanak meg, de más rendszerek (vagy a központi rendszer) felhasználó számára is biztosítanak hozzáférési lehetőséget a helyi adatokhoz.

Elosztottság

- Míg az autonómia tulajdonság a vezérlést jellemezi, addig az adatok térbeli elhelyezkedésére az elosztottság utal. Az elosztatlan, lokálisan működő rendszerek mellett még két nagy csoport különböztethető meg, a kliens-szerver típusú és az egyenrangú elosztás.
- A kliens-szerver modellben a feladatok megoszlanak a kétféle típusú feldolgozó egység között. A szerverek feladata az adatok tárolása és feldolgozása, míg a szerverek szolgáltatásait használó kliensek a felhasználói környezet kialakítását végzik.
- Az egyenrangú modellben a kliens és szerver szerep nincs megkülönböztetve, minden egyes elem rendelkezik az összes funkcióval, és azonos szerepű partnereként kommunikálnak a feladat megoldása érdekében.

Elosztás dimenziói



Tervezési módszerek

- Top-Down
    - Követelményanalízis
    - rendszer globális sémái
    - elosztás tervezés
    - Lokális sémák, fizikai sémák
- Bottom-Up
    - Főkomponensek létrehozása
    - A komponensekből hozzuk létre a rendszert
    - Jó, ha a meglévő elemeket akarjuk integrálni egy rendszerbe

Performancia

Autonóm elosztott rendszerek

Szakterületi modellezés

Szakterületi nyelv

- Szakterületi nyelv (Domain-Specific Language, DSL)
    - Programozási, vagy specifikációs nyelv korlátozott kifejezőerővel, egy konkrét szakterület problémáinak és megoldásainak leírásához.
- Előny: tömör, érthető, kifejező, kódgeneráláshoz jó alap, produktív
- Hátrány: kezdeti költség, betanulás szükséges, elburjánozhat
- Csoportosítás
    - A nyelv jellege szerint
        - Internal DSL
            - Általános célú programozási nyelv speciális módon használva
            - A nyelvelemek közül csak néhányat használunk
            - Szakterületi API
            - Pl. script nyelvek; saját framework hívások
        - External DSL
            - Saját nyelv, nem az alkalmazás programozási nyelve
                - Saját szintaxis, vagy egy más nyelv szintaxisa
            - Pl. a PicProcessor nyelv; Unix parancsok; SQL
    - A programozási mód szerint
        - Deklaratív DSL
            - A végrehajtás módja helyett a kívánt eredmény a fontos
            - Pl. SQL, XAML
        - Imperatív DSL
            - A végrehajtás módját adjuk meg
            - Pl. Workflow nyelv, LOGO

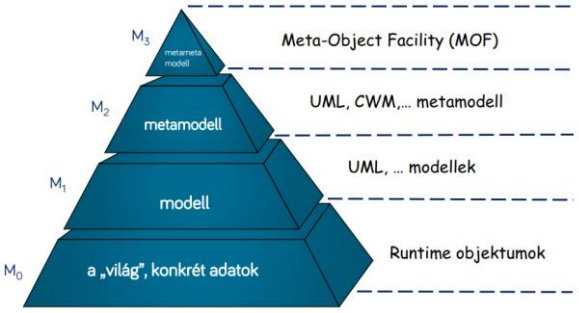
Metamodellezés

- Metamodellezés: A példányosítás reláción alapuló folyamat, ami egy szakterület absztrakt szintaxisát formálisan adja meg.
    - Modell elemek
    - Kapcsolatok az elemek között
    - Attribútumok (elemek és kapcsolatok attribútumai)
    - (Kényszerek és szabályszerűségek)
- Független a konkrét szintaxistól
- Metamodellezés vs …
- Saját megoldás
    - Formális, szabványos automatikus feldolgozás
    - Közös nyelv
- UML profile
    - Nem kötik az UML szabályai
    - Rugalmasabb, testreszabhatóbb
- Blockly

    - Példányosítás vs. sablon-alapú működés
    - KapcsolatokA metamodellező nyelv
- A modellek absztrakt szintaxisát a metamodellek definiálják.
- Mi adja meg a metamodellek absztrakt szintaxisát?
- Metamodellező nyelv
    - Maga is egy (speciális) szakterületi nyelv
    - Metamodellek definiálására használható
    - Definiálja a modellezőnyelvek lehetséges absztrakt szintaxisát

MOF (Meta-Object Facility)

Szint	Leírás	Példák
Metameta-modell	Metamodellezési architektúra Metamodellező nyelvek létrehozására.	MetaClass, MetaAttribute, MetaOperation
Metamodell	Metametamodell példánya. Szakterületi és modellezési nyelvek létrehozására.	Class, Attribute, Operation, Component
Modell	A metamodell példánya, konkrét szakterületi modellek.	Product, Unit Price, Customer, Sale, Detail
Objektumok, adatok	A modell példánya, konkrét, adatokkal kitöltött modell	<Chair>, <Desk>, $100, $200



Interpreter vs. Compiler

- Interpreter
    - Memóriában fut
    - Utasításonként dolgozza fel a programot
    - Gyorsan indul, lassan fut
    - Csak az első hibáig fut jellemzően
    - Debug könnyű
- Compiler
    - Kódot generál
    - Egész programot egyben dolgozza fel
    - Lassan indul, gyorsan fut
    - A végén jelzi a hibákat
    - Debug nehéz

Fordítás

1. Lexikai elemzés
    - Szövegből token
2. Szintaktikai elemzés
    - Nyelvtani szabályok alapján szintaxisfát épít
    - A fa hierarchiába rendezi a tokeneket
3. Szemantikai elemzés
    - Elemzi, ellenőrzi, kiegészíti és átformálja a szintaxisfát
    - Típusellenőrzés
4. Optimalizáció
    - Strength reduction – azonos értelmű, de gyorsabb műveletre cserélés
    - Felhasználatlan, inicializálatlan változók
    - Konstansokon végzett műveletek összevonása
    - Szintaxisfa átstrukturálása
5. Kódgenerálás

Lexikai elemzés

- Cél: bemeneti szövegből tokenek előállítása
- Tagolatlan szöveg -> token sorozat
- Lexéma: a szöveg egy része, logikai egységetalkot
- Token: logikai egység
    - A szöveget a lexéma tartalmazza
    - Van típusa (pl. T_identifier, T_if)
    - Lehetnek attribútumai
    - Egyetlen lexéma – lexéma csoportok
- Összerendelési módszerek:
    - Nyelvtani szabályok
    - Reguláris kifejezések
    - Formális nyelvek
    - Automaták

Szintaxis

- Absztrakt szintaxis:
    - „A nyelvünkben legyen egy „és” operátor, aminek két bemenő paramétere van és bool típusú eredményt ad.”
- Konkrét szintaxis:
    - „Az „és” operátort jelöljük „&&”-el

Szemantikai elemzés

- Szintaktikailag helyes programok mindig helyesek?
    - NEM
    - Változó hivatkozása deklaráció előtt
    - Scope (hatókör) ellenőrzés
    - Típusellenőrzés
    - Kifejezések típusainak ellenőrzése
    - Speciális ellenőrzések:
    - Pl. break, continue csak bizonyos blokkokon belül állhat
- Szemantikai ellenőrzés:
    - A fenti hibák kiküszöbölése
    - Szimbólumtábla készítés – a kódban hivatkozott szimbólumok (változók, osztályok stb.) és azok tulajdonságainak gyűjteménye
- Scope (hatókör): egy entitás hatóköre a program azon helyei, amelyekben az entitás neve az entitásra utal.
- Szimbólumtábla
    - Lépésenként dolgozzuk fel a programot
    - Szimbólumtábla: egy leképezés, amelyben az egyes neveket leképezzük a hozzájuk tartozó entitásokra
    - A szemantikai analízis során folyamatosan karbantartott adatszerkezet
    - A program különböző pontjain más és más a tartalma

Optimalizáció

- Cél: az erőforrások hatékonyabb kihasználása
- Mit optimalizálunk?
- Futási idő, memória, kódméret, …
- A program kimenete nem változhat!
- A fordítóprogramok legkomplexebb része
- Egyensúly: fordítási idő vs. hatékonyság növelése
- Általában IL szinten (gépfüggetlen, de hatékony)

Optimalizáció szintjei, alapblokk

- Alapblokk
  - Egy programban egymást követő utasítások sorozata, ami:
    - Nem tartalmazhat címkét (kivéve az első utasítást)
    - Nem tartalmazhat ugró utasítást (kivéve az utolsó utasítást)
- Szintek:
  - Lokális
    - Alapblokkon belül
  - Globális (nem tanuljuk)
    - Alapblokkok sorozatán
  - Interprocedurális (nem tanuljuk)
    - Metódushatárokon átívelő
    - Ritkán támogatják a fordítók

Optimalizáció eszközei

- Common subexpression
  - Ha
    - Van több értékadás művelet
    - A jobb oldal azonos
    - Jobb oldal résztvevői nem változtak meg
  - Akkor felhasználhatjuk a korábbi eredményt
- Copy Propagation
  - Ha
    - Van egy értékadás, ahol a forrás egy változó
    - Egyik résztvevő értékét sem változtatjuk meg
  - Akkor
    - A két változó kicserélhető
  - Más optimalizációkat segít elő
- Elérhető kifejezések
  - Elérhető kifejezés (available expression): ha van aktuálisan olyan változó, ami a kifejezés értékét tartalmazza
    - Pl: common subexpression: lecserélünk egy elérhető kifejezést az őt tartalmazó változóval
  - Elérhető kifejezések felderítése
    - Kezdetben üres halmaz
    - a=b+c kifejezésnél
      - Az a-t tartalmazó kifejezéseket kivesszük
      - a=b+c kifejezést betesszük
- Dead code
  - Ha
    - Egy értékadás bal oldalán szereplő változó értékét sehol nem olvassuk ki
  - Akkor
    - Az értékadás törölhető, mivel a változó nem módosít a futás eredményén
  - Liveness analízis
    - Dead code felderítés: liveness analysis
    - Egy változó élő (live) a program egy pontján, ha a később következő kódban előbb olvassák ki az értékét legalább egyszer, minthogy felülírnák azt
    - Dead code szűrés ennek megfelelően:
      - Minden változóra liveness számítás
      - Minden olyan értékadás törlése, ami nem élő változónak ad értéket
    - Fordított sorrendben dolgozzuk fel az alapblokk utasításait

        - Néhány változó alapesetben élőnek számít (kimentet befolyásolja pl.)
- Aritmetikai egyszerűsítés
    - Peephole optimalizáció
    - Nehezebben számítható művelet kiváltása egyszerűbbel
    - x = 4 * a; -> x = a << 2;
- Constant folding
    - Fordítási időben kiszámítható műveletek eredményét használjuk a művelet helyett
    - x = 4 * 5; -> x = 20;
- >> 1 = /2
- <<1 = *2

UML Profile

- Az UML modellek egy részét jelöli ki, „jól formáltságot” biztosít a kijelölt tartományban.
- Általános elemek specializációját írja le, sztereotípiákkal, tag-ekkel és kényszerekkel.
- Nem mond ellent az eredeti specifikációnak.
- Visual Studio is támogatja
- Példák:
    - SysML
    - Modeling and Analysis of Real Time and Embedded systems (MARTE)
    - Service oriented architecture Modeling Language (SoaML)
    - UML Profile for Advanced and Integrated Telecommunication Services (TelcoML)
    - UML Testing Profile (UTP)
    - UML Profile for Voice
- Az egyik legnépszerűbb UML Profile: SysML
    - Általános modellező nyelv rendszertervezéshez
    - Szoftver, hardver, információmenedzsment, folyamatok
    - Specifikáció, analízis, tervezés, verifikáció és validáció

OCL (Object Constraint Language)

- Lehetővé teszi precíz metamodellek definiálását
- Tulajdonságai
    - Az OCL kényszerek deklaratívak: azt adják meg mi helyes és nem azt, hogy mit kell tenni
    - Az OCL kényszereknek nincs mellékhatásuk: az OCL kifejezések kiértékelése nem változtatja meg a rendszer állapotát
    - Az OCL kényszerek formális szintaxissal és szemantikával rendelkeznek: értelmezésük egyértelmű és automatizálható
- Kontextus
    - Kontextus: Az a modell elem, amire az OCL kifejezést definiálták
        - osztály, interfész, adattípus, komponens, művelet, példány
    - Kontextus típusa: annak az modellelemnek a típusa, amire a kifejezést kiértékelik
    - Ha a kontextus egy típus, akkor a kontextus típusával megegyezik
    - Kontextus példány: a konkrét modellelem, amire kiértékeljük a kifejezést
        - „self” kulcsszóval hivatkozzuk context Customer inv: self.name = ’Edward’
- Kifejezéstípusok kontextus szerint
    - Invariáns
        - Metamodellelemre definiált kényszer
        - Egy logikai kifejezés, aminek a metamodellelem minden példányára, minden időpillanatban igaznak kell lennie
        - „A felnőtt ember kora nagyobb, mint 18”
    - Elő- és utófeltétel

        - Műveletre definiált kényszer
        - Előfeltétel: egy adott művelet elvégzése előtti utolsó időpillanatban igaz feltétel
        - Utófeltétel: egy adott művelet elvégzése utáni első időpillanatban igaz feltétel
        - „A túra elején 3 kiló étel volt nálunk, a végén 0”
    - Kezdeti értékek
        - Attribútumra vagy asszociációra definiált kényszer
        - Egy érték, amit az attribútum vagy asszociáció vesz fel a kontextus példány létrejöttének pillanatában
        - „Születéskor minden gyerek szeme kék”
    - Származtatott érték
        - Attribútumra vagy asszociációra definiált kényszer
        - A származtatott elem nem önmagában létező érték, mindig más elemek segítségével definiálják
        - „A végső érdemjegy a ZH és a vizsga átlaga”
    - Metódustörzs
        - Műveletre definiált kényszer
        - Vannak olyan műveletek, melyek egy adott értéket kérdeznek le, nincs egyéb mellékhatásuk
        - Kényszerben rögzíthetjük, hogy pontosan mit is kell visszaadniuk
        - “A könyvtár összes könyve a polcokon lévő könyvek számának összege”
- Invariánsok
    - Metamodellelemre definiált kényszer
    - Egy logikai kifejezés, aminek a metamodellelem minden példányára, minden időpillanatban igaznak kell lennie
- Öröklés
    - A kényszerek is öröklődnek
    - Az invariánst a származott osztály örökli. A származott osztály szigoríthatja a kényszert, de nem enyhítheti.
    - Az előfeltételt lehet enyhíteni a származott osztály újradefiniált műveletében. Szigorítani nem lehet.
    - Az utófeltételt lehet szigorítani a származott osztály újradefiniált műveletében. Enyhíteni nem lehet.
    - Műveletek örökléssel:
        - OclIsTypeOf(típus)
            - Igaz, ha a modellelem típusa megegyezik az argumentumbelivel
        - OclIsKindOf(típus)
            - Igaz, ha vagy megegyezik az argumentumbeli típussal, vagy valamelyik ősének típusával
        - OclAsType(típus)
        - Castolás az adott típusra. Ha nem kompatibilis a típus, akkor OclInvalid-ot ad vissza
- Navigációs kifejezések
    - Az asszociációvégek (role name) navigálásra használhatóak az egyik objektumból a másikba
    - A navigációkat attribútumként kezeljük
    - A navigációs kifejezés típusa lehet:
    - Felhasználó által megadott típus (az asszociáció végének multiplicitása 0..1)
    - Gyűjtemény (az asszociáció végének multiplicitása > 1)

Vizuális szakterületi nyelv

- Absztrakt szintaxis
    - Nyelv struktúrája
    - Kiegészítő kényszerek
- Konkrét szintaxis
    - Megjelenítés

- Szemantika
    - Struktúra megjelenítése

Konkrét szintaxis grafikus szakterületi nyelvek esetén

- Grafikus modellekkel sok minden egyszerűbb
    - Kisebb a betanulási idő
    - Könnyebb és gyorsabb az információátadás
- Konkrét szintaxis
    - Klasszikus programozási nyelvek:
        - Konkrét szintaxis: amit a programozó lát, a nyelv szintaktikája
        - Absztrakt szintaxis: amit a fordító lát, a nyelv absztrakt reprezentációja
    - Vizuális szakterületi nyelvek:
        - Konkrét szintaxis: a szakterületi függő megjelenítés definíciója
        - Absztrakt szintaxis: a nyelv absztrakt jelentésének definíciója (amit a metamodellel adunk meg általában)
- Szöveges DSL – konkrét szintaxis
    - A nyelv, ahogy megjelenik a programozó/felhasználó számára maga a konkrét szintaxis
    - A szöveges DSL definíció jellemzően magába foglalja a nyelv konkrét szintaxisát
    - Szerkesztőeszköz szinten a konkrét szintaxis egyes elemei kiemelhetőek (syntax highlight)
- Vizuális DSL-ek konkrét szintaxisa
    - Konkrét szintaxis megadási módszerek vizuális DSL esetén
        - Nincs DSL-függő konkrét szintaxis
        - Metamodell kiegészítések
        - Metamodell + modell kiegészítések
        - Kódolás
        - Konkrét szintaxis nyelv modellek
- Szakterület független megjelenítés
    - Nincs DSL-függő konkrét szintaxis
        - Alapértelmezett megjelenítés
        - Egységes
        - Nem kell kifejleszteni
        - Elég absztrakt
        - Többségében osztálydiagram-szerű
        - Prototípus tesztelés, metamodellek leírása
- Metamodell kiegészítések
    - Metamodell kiegészítése megjelenítéssel kapcsolatos attribútumok (ikon, méret, betűtípus)
        - Azonos metamodellt követő modellek könnyen használhatóak
        - Azonos metamodell -> egyfajta megjelenítés
        - Modellező keretrendszernek támogatnia kell a speciális attribútumokat
        - Vizualizáció és adat keveredik
- Metamodell + modell kiegészítések
    - Megjelenítés leíró attribútumok 2.
        - Metamodellben és modellben is lehet definiálni
        - Egy metamodellhez több kinézet is tartozhat
        - Modell szintű attribútumok miatt a metamodellt meg kell változtatni
        - Vizualizáció és adat keveredik
- Konkrét szintaxis forráskóddal
    - Kódolás
        - Metamodell, vagy modell alapon
        - Szakterületenként elég egyszer

        - Testreszabható, rugalmas
        - A metamodell nem változik
        - Adat és vizualizáció elválik egymástól
        - Szakértő nem ért hozzá (nem tud kódolni)
        - Lassú
        - Potenciális hibaforrás
- Konkrét szintaxis forráskóddal
- Konkrét szintaxis modellezése
    - Konkrét szintaxis modell
    - A kinézetet írja le, speciális nyelv
    - A metamodellhez kapcsolható
    - Nem igényel külön kódolást (Felhasználóbarát)
    - Egy metamodellhez több kinézetet is meg lehet adni
    - Kinézetet korlátozhatja a leíró nyelv
    - Interakció (állapotok) modellezése nehéz
- Konkrét szintaxis megadási módszerek
    - Metamodell – konkrét szintaxis összekapcsolása
        - pl. Külön leképzési modell (Eclipse GMF)Konkrét szintaxis megadási módszerek
    - Metamodell – konkrét szintaxis összekapcsolása
        - Konkrét szintaxis része a leképzés (VMTS)
- Konkrét szintaxis a gyakorlatban
    - GME – Ikonok, címkék
    - MetaEdit+ – Saját grafikus nyelv (konkrét szintaxis modellhez hasonló, varázslón alapuló megoldás)
    - Eclipse – Forráskód / Konkrét szintaxis nyelv
    - VMTS – WPF-alapú pluginok / Konkrét szintaxis nyelv

Reguláris kifejezések

- ?
    - A megelőző minta 0 vagy 1 alkalommal fordul elő.
- +
    - A megelőző minta 1 vagy több alkalommal fordul elő.
- *
    - A megelőző minta 0 vagy több alkalommal fordul elő.
- {m,n}
    - Segítségével megadható minimum és maximum vagy pontosan megadott számú előfordulás - {3} pontosan 3 előfordulás; {3,} legalább 3 előfordulás; {2,5} legalább 2 legfeljebb 5 előfordulás; {,10} legfeljebb 10 előfordulás. A d.{,5}ány igaz minden esetben, ha legfeljebb 5 karaktert kell helyettesíteni, például a dolmány esetén.
- ^
    - A minta eleje: Ezzel jelezhetjük, hogy a kifejezést a minta elején keressük.
- $
    - A minta vége: Az előző horgonyhoz hasonlóan a minta végét testesíti meg.
- |
    - Vagy.
- ( )
    - Kifejezések csoportosítása: Nem csak a vagylagos egyezés az egyetlen lehetséges felhasználás.
- . (pont)
    - Bármilyen karakter
- [karakterek]
- [^karakterek]

- A kapcsoszárójelek között felsorolt karakterek egyikével sem egyező karakter.

Nyelvtan és mintaszöveg

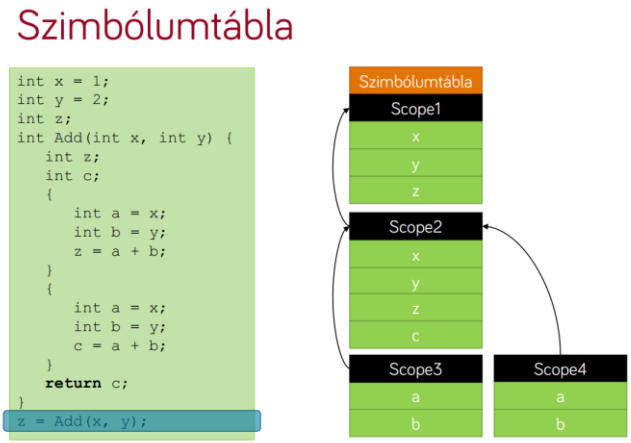
```
E →T | Z | T + (E) | E + T | E^T
Z →EE
T →0|1|…|9


5 + (12^3 + 1)
```

- Megoldás:

E --> T + (E) --> T + (E + T) --> T + (E + 1) --> T + (E^T + 1) --> T + (E^3 + 1) --> T + (Z^3 + 1) → T + (EE^3 + 1) → T + (ET^3 + 1) --> T + (E2^3 + 1) --> T + (T2^3 + 1) --> T + (12^3 + 1) --> 5 + (12^3 + 1)

- Szabályok
  - Minden sorban egy változtatás
  - Illeszkednie kell a típusra és számra

Linkelt szimbólumtábla



Szemantika

- Szemantika: a nyelv/modell jelentése valamilyen módszert, formalizmust követve.
- Szemantika használat előnyei:
  - Gépi feldolgozás megkönnyítése
  - Eszköz független specifikáció és viselkedés vizsgálat
  - Fordítók és generátorok ellenőrzésének lehetősége
  - Optimalizáció, átalakítás utáni ekivalencia
  - Bizonyítások megkönnyítése (automatikus és manuális esetben is)
  - Nyelvi redundancia elkerülése, források eltérő fogalmai közti megfeleltetés
- Szemantika megadási módszerek
  - Informális
    - ~ Pontos, szöveges specifikáció
  - Operációs (hogyan)
    - A működés módjára fókuszál
  - Denotációs (eredmény)
    - A működés eredményére fókuszál
  - Axiomatikus (tulajdonságok)
    - Axiomákból vezeti le a működést (matematikai) logikával
    - Elő és utófeltételek
  - Algebrai szemantika, transzlációs szemantika, ….

- Informális szemantika
    - Peano
    - Változócsere
        - Az utasításokat a ‘;’ szimbólum választja el egymástól, ez a ‘;’ jelentése
        - Egy utasítás egy változóval kezdődik, amit egy ‘:=’ szimbólum és egy kifejezés követ
        - A művelet során a változó értéke a ‘:=’ szimbólum utáni kifejezés értékét veszi fel
    - Decision Node
        - Az activity diagram szemantikáját a token áramlás adja meg. Egy token végighalad egy élen: leveszik a forráscsúcsról, és átteszik a célcsúcsra. A token indítja el az adott csúcs végrehajtását.
        - Tokeneket nem duplikálunk.
        - Minden token, mely egy decision node-ra érkezik, pontosan egy kivezető élen megy tovább
- Operációs szemantika
    - Tulajdonságok
        - Hogyan kell végrehajtani
        - Milyen műveletek, milyen sorrendben
        - Strukturális (“small step”) vs Természetes (“big step”)
        - Általában két részből áll:
            - Nyelv – a modell, program nyelve
            - Interpreter (értelmező)
                - Modell feldolgozása = elemi lépések legenerálása
                - Absztrakt gép, vagy rekurzív fv.
            - Főként szekvenciális nyelveknél
    - Peano
    - Változóscsere
        - A szövegrészlet utasítások sorozata, melyeket a ‘;’ szimbólum határol.
        - Az utasításokat balról jobbra hajtjuk végre
        - A x:=y utasítás jelentése az, hogy először kiértékeljük y kifejezést, majd ezt értékül adjuk x változónak
    - Activity Diagram
- Denotációs szemantika
    - Tulajdonságok
        - A művelet hatására koncentrál
        - Információ denotációja: Leképzés matematikai reprezentációra, ahol a pontos jelentés ismert
            - 10+3 denotációja 13, ami egy természetes szám
        - Három részből áll:
            - Nyelv (a cél szakterület)
            - Szemantikai algebra (számítási modell leírására)
            - Leképző függvény
        - Kompozicionalitás
        - 0. példa, függvény ([0..10]-> N) szemantika
            - Bemenet – kimenet párok felsorolása táblázatban
            - F{[0 0] [1 1] [2 1] [3 2] [4 3] [6 8]}
            - Bemenet – csak az értelmezett bemenetek
        - Többnyire a programozási nyelvek elméleti és összehasonlító elemzéséhez használjuk
        - Matematikai objektumokkal foglalkozunk, nem implementálunk
            - Így könnyebb a programokról tulajdonságokat belátni, pl.:
                - Egy érték mindig konstansra értékelődik ki
                - Minden változót inicializáltunk-e induláskor

        - Minden része elérhető egy programnak
    - Peano
    - Változócsere
        - A ‘;’ szimbólummal elválasztott utasítások halmaza az egyes utasítások függvénykompozíciója
        - Az az utasítás, ahol egy változót a ‘:=’ szimbólum és egy másik változó követ, egy függvény
    - Activity
        - Az activity diagramot leképezzük egy folyamatalgebrára
        - Kompozicionalitás
- Axiomatikus szemantika
    - Tulajdonságok
        - Nem egy teljes program, vagy modell jelentését megadni
        - Axiómák
        - Összetett kifejezések visszavezetése az axiómákra
        - A műveletek helyességének ellenőrzésére szolgál
        - Matematikai logikán alapul (Hoare logika)
            - {P} c {Q}
            - Ha a program kielégíti az előfeltételeket a futás előtt, akkor biztosítja, hogy a futás végén az utófeltételek is ki lesznek elégítve
            - Ha az előfeltétel nem teljesül nem állítunk semmit csak részleges bizonyítása
        - Feltételek ellenőrzése, bizonyítások (terminálás, dead lock, stb.)

Szimuláció

- Szimuláció ha,
    - …ha a fizikai rendszert lehetetlen megépíteni…
        - Galaxisok ütközésének szimulálása
    - … vagy költséges…
        - Repülőgép légáramlás szimuláció
        - Szabályozási rendszerek
    - … vagy nem megfigyelhető…
        - Predikció, pl. vírus terjedés
    - … vagy mérési adat-szegény a környezet
        - Pl. közlekedési lámpák beállítása
            - Minden egyes kipróbált konfiguráció áteresztőképességének mérése sokáig tart
- Szimuláció céljai
    - Analízis
        - Az analizálásra kerülő rendszer vagy létezik, vagy tervezés alatt áll. Megpróbáljuk megérteni a viselkedési karakterisztikáját
    - Következtetés
        - A rendszer létezik. Megfigyelésekből próbáljuk a viselkedésére következtetni.
    - Tervezés
        - A rendszer tervezés alatt áll, olyan formában ahogy modellezzük. Szeretnénk minél jobb tervet készíteni
- Szimulátor
    - (Informatikai) Rendszer, mely képes a modellt végrehajtani, hogy a rendszer viselkedését utánozza
    - Különbözik a modelltől
    - Egy modellhez több szimulátor is tartozhat
    - Szimulációs algoritmusok formálisan megadhatók, ami elősegíti a verifikációt

Modellezési relációk

- Érvényesség
    - Egy modell hűen fejezi ki a rendszert a megadott kísérleti keretben
- Replikatív érvényesség
    - Az összes lehetséges, már ismert eredményű kísérletre a modell és a rendszer viselkedése megegyezik (elfogadható hibahatáron belül)
- Prediktív érvényesség
    - Lehetséges a még nem ismert viselkedés előrejelzése
- Strukturális érvényesség
    - Nem csak egészében érvényes, hanem állapotonként vagy komponensenként is

Szimuláció formalizmusok

- DTSS: diszkrét idejű rendszerek
    - Lépésenkénti végrehajtás
        - Digitális óra, időegységek
        - Időegység tetszőleges (1 mp, 1 perc, …)
    - Egy időegységben a modell egy állapotban van
    - Példa: Sejtautomata
        - Adott egy 2 dimenziós rács (diszkrét tér)
            - Minden elem egy sejt, melynek állapota a szomszédjaitól függ (átlós elem is szomszéd)
            - Állapotváltozás vizsgálat az idő léptetésekor
        - Szabályok:
            - Egy sejt életben marad, ha 2 vagy 3 élő sejt van a szomszédságában
            - Háromnál több (tumultus) vagy kettőnél kevesebb szomszédnál (izoláció) elpusztul
            - Halott sejt életre kell, ha pontosan 3 szomszédja van
        - Viselkedés típusok
            - A minta hamar kihal
            - A minta periodikussá válik
            - Kaotikus viselkedés
            - Megjósolhatatlan, nem periodikus, de szabályos mintákat mutat
        - Példaviselkedések
            - Tartós minták
            - Oszcilláló minta
            - Mozgó minta
- DESS: differenciál egyenletes rendszerek
    - Nincsenek időegységek, az idő folyamatos
    - Állapotátmeneti függvény helyett származtatási függvény
    - Nem a következő állapotot adjuk meg, hanem a változás mértékét, ebből kell kiszámítani a következő állapotokat
- DEVS: diszkrét eseményű rendszerek
    - Példa: gyártási folyamat
        - Nyersanyagok megrendelése
        - Nyersanyag tárolása
        - Termék gyártása
        - Minőségbiztosítás
        - A gyártási folyamat jellemzően esemény vezérelt
            - Rendelés alacsony raktárkészletnél
            - Minőség ellenőrzése gyártás után
            - Termék visszavétele ha hibás
            - Raktárba szállítás ha jó

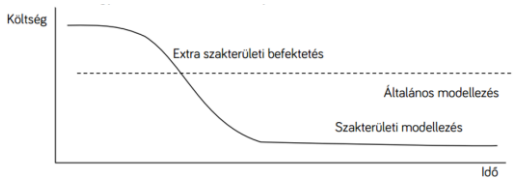
        - DTSS és DESS módszerek alkalmazhatóak, de nem ideálisak a feladat elvégzésére
- Informális definíció
    - A rendszerek következő tulajdonságait jegyezzük meg:
    - Egy diszkrét rendszer komponensekből áll, melyek együttműködnek a kívánt viselkedés elérése érdekében
    - A diszkrét rendszer komponensei üzenetek küldésével és fogadásával kommunikálnak, amelyeket eseményeknek hívunk
    - A rendszer különféle állapotokkal rendelkezik, amelyek diszkrétek. Az állapotváltás események hatására történik.

Modeltranszformációk típusai

- A bemenetek és kimentek száma (In-place vs out-place)
- A nyelv szerint (Endogenous vs exogenous)
- Az irány szerint (unidirectional vs bidirectional)

Generatív programozás

- Programozási módszertan, alapja az automatikus forráskód-generálás
- Párhuzamba vonható a komponens-alapú szoftverfejlesztéssel és a termékcsalád tervezéssel
- Újrahasznosítható termék
- Nem egyszeri fejlesztés, folyamatos evolúció

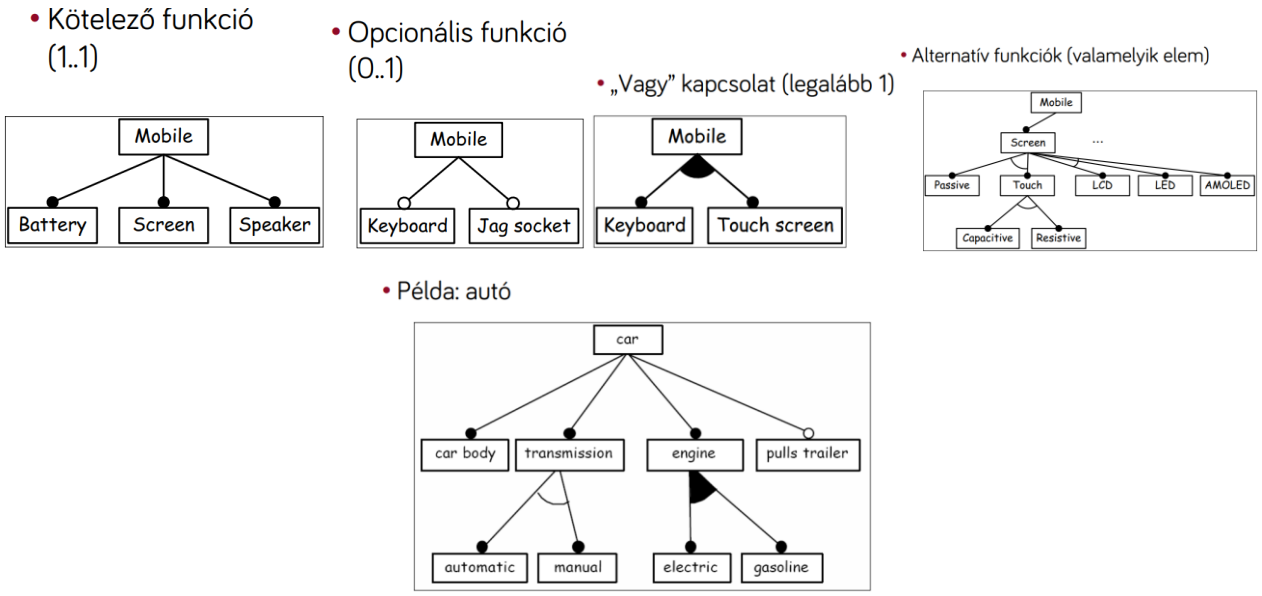


- Generatív paradigma
    - Működés: modellezőnyelv+generátorok
    - Többszöri használatnál éri meg
- Kódgenerálás
    - Nincsenek univerzális DSL fordítók (mint C fordítók)
    - Gyakran a DSL-t és a generátort ugyanott fejlesztik
    - Gyors fejlesztés, finomhangolási lehetőség
    - Hibalehetőségek
- Alkalmazás generálása
    - Amire érdemes figyelni
    - Túl részletes/általános nyelv => kicsi absztrakciós szint ugrás => kis előny generátorból
    - DSL – szakterület nem illeszkedik => komplex generátor (validálás + hozzáadott infó miatt)
        - Jel: fejlesztők úgy építik a modellt, hogy a generátor elfogadja

Funkciómodellezés

- Implementációtól független, tömör leírása a különböző szakterületi változatoknak
- Kulcs: újrahasznosítás
- Segít elkerülni:
    - Fontos funkció / variáció kimaradjon
    - Feleslegesen vegyünk fel funkciót / variációt
- A funkciómodell általában a konkrét termékpéldányok közti különbséget ábrázolja
- A funkciómodell (feature diagram) a termékcsalád konfigurációs lehetőségeit ábrázolja
    - Modellelemek
        - Csomópontok

        - Irányított élek
        - Jelölés az éleknél
    - Gyökérelem: fogalom (concept)
    - Funkciók (feature node)
- Központban a funkció (feature)
    - Tudás egy része, a fogalom építőeleme
    - Segít megtalálni az azonosságokat, különbségeket termékek, termékcsoportok között
    - Hasznos ha több változat van ugyanabból a termékből



Modellvezérelt architektúra (MDA)

- A modellvezérelt architektúra alkalmazásakor a rendszerfejlesztés fázisai
    - Modellezési fázis:
        - platform-független, a rendszer funkcionalitását írjuk le
        - Platform-Independent Model – PIM
    - A rendszer erőforrásainak allokálása
        - Model Processor – MP
        - Platform-Specific Model – PSM
    - A rendszer implementációjának forráskódja automatikusan generálható
        - PSM alapján platformfüggő kód

Modellfeldolgozási módszerek

- Feldolgozási módszerek
    - Bejárás-alapú
    - Sablon-alapú
    - Gráftranszformáció-alapú
    - Vegyes
- Bejárás-alapú feldolgozás
    - A modellnek van API-ja
        - Csomópontok: objektumok
        - Kapcsolatok: referenciák, referencia kollekciók
    - A feldolgozást valamilyen imperatív nyelvvel algoritmizáljuk
        - Lehet teljesen saját megoldás
            - Tipikusan in-place (input model = output model)

        - Szöveges output
    - Erősen feladatfüggő, nincs szabvány rá
- Sablon-alapú feldoglozás
    - Tipikusan forráskód vagy más szöveges output generálására
    - Adott a szöveges kimenet váza
        - A hiányzó részeket kiegészítjük
            - XSLT

XSLT

- EXtensible Stylesheet Language Transformations
- Feldolgozás, sablonok (template) illesztésével
- XML dokumentumok transzformációja
    - Deklaratív szemantika (XML)
    - XML vagy más, tetszőleges szöveges kimenet
    - Navigáció XPath-szal

Acceleo

- Kódgenerátor
- OMG MOF Model to Text Language (MTL) implementáció
- Eclipse alapú
- Két fő rész
    - Sablonok -> kódgenerálás
    - Lekérdezések (OCL alapon) -> információ lekérdezés
- Sablonok (templates)
    - Előfeltételek (mikor fusson le a sablon?
    - Utófeltételek (kódformázás)
    - Változó inicializáció
- Lekérdezések (queries)
    - Java kódrészletek megengedettek
    - Ciklusok, feltételes elágazások, értékadás

Microsoft T4

- Text Templating Transformation Toolkit
- Szöveg blokkok és vezérlési logika egy fájlban
    - Szöveg blokk kimásolódik a kimenetre
    - C# vagy VB
        - Tud írni a kimenetre
    - Hasonló, mint az ASP.NET, PHP, …
- Ahol használják: DSL Tools, Entity Framework, VMTS...
- T4 fejléc
    - Kötelező: a template nyelv
    - assembly
        - Alapértelmezetten használható az mscorlib
    - Import
        - Alapértelmezetten használható a System

Xtend

- Általános célú programozási nyelv
- Objektum orientált
- Transzparens együttműködés Java-val

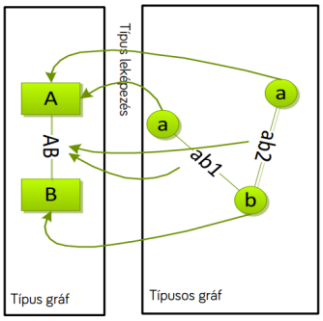
    - Statikus típusellenőrzés
    - Java típusrendszer
    - Java kódra fordul
    - Oda-vissza hivatkozás
- Modernizált Java

Modelltranszformációk összefoglalva

- Elemi lépések (szabályok)
    - Az input modell egy részének kijelölése
    - A kijelölt rész módosítása: törlés, hozzáadás
- Elemi lépések sorrendjét rögzítjük (pl. vezérlésfolyamgráf)
- Vizualizáció -> könnyebb érthetőség (ld. szakterületi modellezés)
- Formális háttér (gráfújraírás) -> verifikációs lehetőségek
- Attribútumok kezelése
    - A gráf definíció kiegészíthető attribútum specifikációval
    - A gráftranszformáció alapvetően a struktúrát módosítja
    - Az attribútumokat általában hagyományos kóddal kezeljük
- Speciális elemekből építkező program
    - Minden megvalósítható?
    - Mikor érdemes használni?

Gráfok

- Alapvető formalizmus:
    - csomópontok, élek,
    - ezekhez attribútumokat rendelhetünk
- Mire használhatjuk ezt a struktúrát a szoftverfejlesztésben?
    - Diagramok (a modellek összetett struktúrájának intuitív megjelenítése) formalizálására
    - Modellek formalizálására
- Típusos gráf
    - Jól leírhatók velük
        - UML
        - Metamodell, - modell kapcsolatok



- Gráftranszformáció
    - Gráftranszformáció
        Eegy gráf módosításának matematikai leírása
        - A gráfokkal leírt modellek feldolgozásának egy módszere, ami a gráfújraíráson alapul
    - Újraírási szabály:
        - Egy gráf módosításának elemi lépése
            - Megkeresünk egy adott mintát az input gráfban (illesztés)
            - És azt módosítjuk (csomópontok, élek létrehozása, törlése)
    - Vezérlési szerkezet

        - Az egyes szabályok alkalmazásának sorrendjét rögzíti, pl:
            - Először az 1-es szabály, utána a 2-es szabály, vagy
            - Alkalmazz egy véletlenszerűen kiválasztott szabályt
- Szabály
    - Illesztés (match):
        - Leírunk egy (rész)gráfot (minta)
        - Ezt megkeressük a bemeneti gráfban
    - Módosítás
        - Élek, csomópontok törlése, létrehozása
    - Egyszer alkalmazzuk a szabályt
    - A minta megtalálásának módjával nem foglalkozunk

Vezérlési szerkezetek

- A szabályok (nem feltétlenül determinisztikus) végrehajtásának sorrendjét specifikálják:
    - Prioritási szinteken alapuló
    - Vezérlésfolyam gráf
- Prioritás szintek
    - A szabályokat csoportokba (szintek) soroljuk
    - Egy szinten tetszőleges sorrendben alkalmazzuk a szabályokat
    - Kimerítő alkalmazás
    - Amikor nincs több alkalmazható szabály továbblépünk a következő szintre
- Vezérlésfolyam gráf
    - Explicit vezérlés
    - Elágazások a szabály sikerességétől függően
    - Kimerítő szabályalkalmazás (egy szabályt addig alkalmazunk, amíg csak lehet)
    - Kezdőpont, végpont
    - Szabály alkalmazás
        - Kimerítő alkalmazás: a szabályt addig alkalmazzuk ismételten, amíg lehet
        - A szabály alkalmazása sikertelen: ha nem sikerült az illesztés, egyébként sikeres
    - Szekvenciális végrehajtás
    - Elágazás egy szabály sikerességétől függően
        - sikeres, ha legalább egyszer sikerült alkalmazni
    - Az eddigiek a legáltalánosabb elemek
    - Konkrét keretrendszerekben kiegészíthetők pl.
        - OCL kényszeren alapuló elágazási feltételekkel
        - Hierarchikus szabályokkal
        - Storage

Transzformációk

- Determinisztikusság
    - Definíció szerint a szabályok illesztése nem determinisztikus, vagyis a keretrendszer feladata, hogy keressen egy lehetséges illesztést
    - Ez ellensúlyozható a szabályok sorrendjének specifikálásával
- Terminálás
    - A transzformáció be kell fejeződjön
- “Meta” transzformációk
    - A transzformációs szabályok leírása egy DSL-el történik
    - A gráftranszformáció jól alkalmazható DSL-eken
    - Ötlet: transzformációk transzformációja
        - (Higher Order Transformation)