XML technikák II

Kovács, László

XML technikák II

Kovács, László

Kelet-Magyarországi Informatika Tananyag Tárház

Nemzeti Fejlesztési Ügynökség http://ujszechenyiterv.gov.hu/ *06 40 638-638*

Lektor

Dr. Johanyák Zsolt Csaba

Kecskeméti Főiskola, főiskolai docens.

A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-Magyarországi Informatika Tananyag Tárház projekt keretében valósult meg.

Tartalom

1. fejezet - XSLT nyelv elemei

1. Az XSLT nyelv áttekintése

Az XML formátum az általánosságából következően rendkívül széles alkalmazási területtel bír. Az XML alkalmas arra, hogy adatbázisként szolgáljon, paraméter adatokat tároljon, dokumentumokat vagy programokat írjon le. Például egy kis cég ügyfél nyilvántartó rendszerében XML szolgálhat az ügyfelek adatainak nyilvántartására. Emellett XML formátumban tárolhatjuk az ügyfeleknek szóló körlevelek sémáit, majd XML lesz az elkészített levelek tárolási formátuma is. Ha ezt az egyszerű ügyviteli folyamatot vizsgáljuk, akkor azt látjuk, hogy az ügyviteli folyamat teljes vertikumában az XML adattárolási formátum jelenik meg. Ez azt is jelenti, hogy az ügyviteli folyamatok lefutása, az információáramlás során az egyik XML dokumentumban tárolt adatok átalakulnak egy másik XML formátum adataivá. Például egy körlevél tartalmát leíró XML dokumentumból, melyben nem szerepelnek formátum leíró elemek, egy teljes körlevelet megadó XML dokumentumot állítunk elő. Az XML formátum dominanciájából eredően megjelent az igény az XML dokumentumok tartalmának hatékony konverziójára is, amikor egy forrás XML dokumentumból egy cél XML dokumentumot állítunk elő.

Habár a megismert programozási technikák, mint például a DOM modell, alkalmas lehet ilyen konverzió elvégzésére, az egyedi procedurális konverzió kódolás ellen a nagy járulékos költség szól. Hiszen minden procedurális programfejlesztési munkánál igen jelentős rész a validálási, ellenőrzési fázis. Másrészt egy alacsony szintű leírást sokkal nehezebb módosítani, továbbfejleszteni a program életének későbbi fázisaiban. Emiatt célszerűbb egy magasabb szintű, imperatív parancsnyelv használata, amelynél külön előnyt jelent az XML formátumra való illeszkedés. A közös XML platform előnyt jelentett az XML Schema-nál is a DTD-vel szemben. A kialakított XML dokumentum konverziós nyelv az XSLT elnevezést kapta, mely az Extensible Stylesheet Transformation Language (Bővíthető Stílus leírás Transzformációs Nyelve) elnevezésből származik.

Az XSLT nyelv első verziója 1999-ben jelent meg a Word Wide Web Consortium (W3C) égisze alatt. A nyelv kialakításának fő hajtóereje az a tapasztalat volt, hogy a web-es megjelenítésnél sok esetben ugyanazt a dokumentumot több különböző formátumban is meg kell jeleníteni a felhasználási környezettől függően. Egy névsort, táblázatot különböző alakban kell előállítani, attól függően, hogy milyen megjelenítő eszközt és keretrendszert használunk. Egész más formátum kell például egy böngészőben vagy egy nyomtatandó jelentésben. A fejlesztés egyszerűsítésének egyik fontos lépése volt, amikor tudatossá vált a tartalom és a megjelenítési forma szeparálhatósága, függetlensége. Ezen törekvés első lépései a stíluslapok (CSS) használatához kötődnek. A CSS lapok segítségével egy adott XML, HTML dokumentumnál szabályozni lehet a megjelenítési paramétereket, többek között a karakterek betűtípusát és méretét. A CSS lapok igen hamar népszerűvé váltak, s a használat során újabb igények léptek fel, amelyek már túlmutattak a CSS lehetőségein. Ugyanis a CSS mechanizmussal csak a megjelenítési paramétereket lehet beállítani, a tartalom módosítására nincs lehetőség. Sok esetben viszont jó lenne a tartalmat is származtatni egy adatsorból. Például egy bajnokság eredménylistáját mint adatsort véve, a kapcsolódó megjelenítés lehet többek között

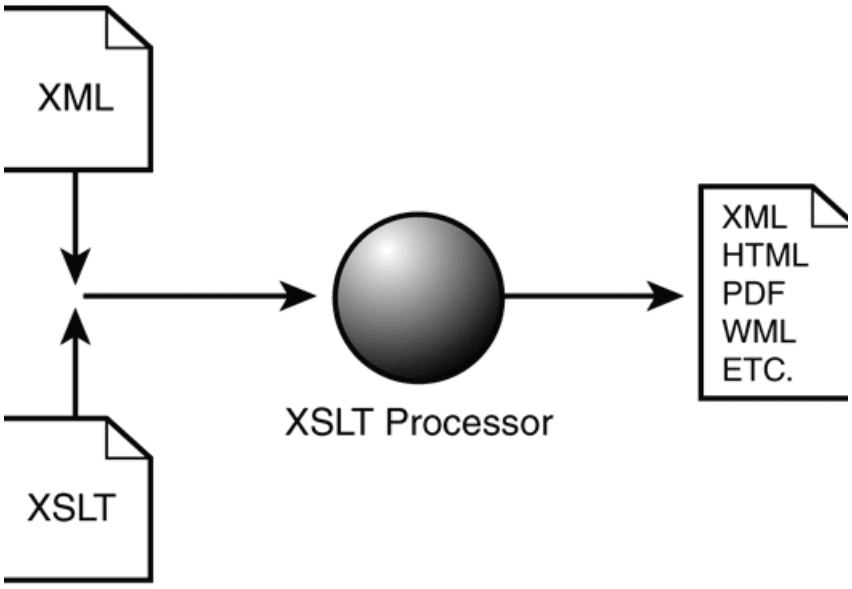
- eredmények felsorolása (a tartalom nem módosul),
- kiválasztott csapat eredményeinek megadása (szűrés),
- ponttáblázat megadása (pontok számítása),
- bajnokság állásának meghatározása (aggregáció és rendezés).

A tartalom módosítására irányuló konverziós nyelvként jött létre a XSLT nyelv. Az XSLT rövid jellemzéseként azt mondhatjuk, hogy XML dokumentumot XML dokumentumba konvertál át. A konverzió menetét egy parancs állományban adjuk meg, amely szintén XML formátumú. Az XSLT-hez kapcsolódó állományok:

- forrás XML dokumentum,
- a transzformációt leíró dokumentum (rendszerint XSL kiterjesztéssel),
- eredményt tartalmazó XML dokumentum.

Az XSLT transzformációs nyelvet imperatív nyelvnek lehet nevezni, hasonlóan az SQL nyelvhez, mivel nem elemi algoritmusokat kell kidolgozni a transzformáció megadására, hanem magasabb szintű mintákkal lehet

dolgozni. Az XSLT nyelvet sokan a funkcionális nyelvekhez sorolják, mivel ott is hasonló elveken nyugszik a kódolás: mintákat definiálunk a programban. Az XSLT program tehát alapvetően minták listája, ahol minden minta megadja, hogy egy adott dokumentumrészt milyen más alakra kell transzformálni. A minták célja a dokumentum érintett részeinek kijelölése.



1. Az XSLT forrás és eredmény objektumai

A dokumentum kezelésekor az XSLT nyelv a dokumentumot nem karakter sorozatként, hanem dokumentum faként kezeli. Ebben az értelemben az XSLT adatszemlélete az XDM modellen alapul. Emiatt nagyban hasonlít a DOM szemléletére is, van azonban néhány apró eltérés a kétféle fa értelmezés között:

- az XSLT-fa egyszerűbb, kevesebb részletre tér ki,
- a DOM fa módosítható, az XSLT fa csak olvasásra vagy csak írásra szolgál,
- az XSLT-fa kezelésnél egyszerűbb szinkronizálós feladatokra van szükség.

Habár az XSLT-fa kezelése számos egyedi vonással is rendelkezik, melyek miatt a hatékonysági szempontokra tekintettel saját XSLT-fa kezelő motort kellett kidolgozni, sok esetben az XSLT feldolgozók a SAX és DOM modellek kezelési mechanizmusára hagyatkoznak, tehát ezek a szabvány modellek lefedik az XSLT modell funkcionalitási igényeit.

Az XSLT minta illesztése során tehát a fában kell meghatározni az érintett csomópontok halmazát. A feldolgozandó elemek kijelölésének rugalmasnak és funkcionálisan gazdagnak kell lennie, hiszen

- a fában több irányban is lehet haladni,
- az elemek mellett más csomópont típusok is vannak,
- egy mintával tetszőleges csomóponthalmazt le lehessen fedni,
- lehetőséget kell adni a tartalom és struktúra alapú szelekcióra,
- egyszerű számítások elvégzését is biztosítani kell.

Mindezen követelmények egy jól megtervezett, hatékony mintaillesztési mechanizmust igényelnek. Mivel ezt a mechanizmust az XSLT mellett más XML szabvány is hasznosítani tudja, erre a célra egy külön XML-kapcsolódó szabvány jött létre, a korábban már megismert XPath szabvány.

Az XSLT használatához, kipróbálásához három elemre van szükség:

- forrás XML állomány,
- transzformációt leíró XSLT állomány,

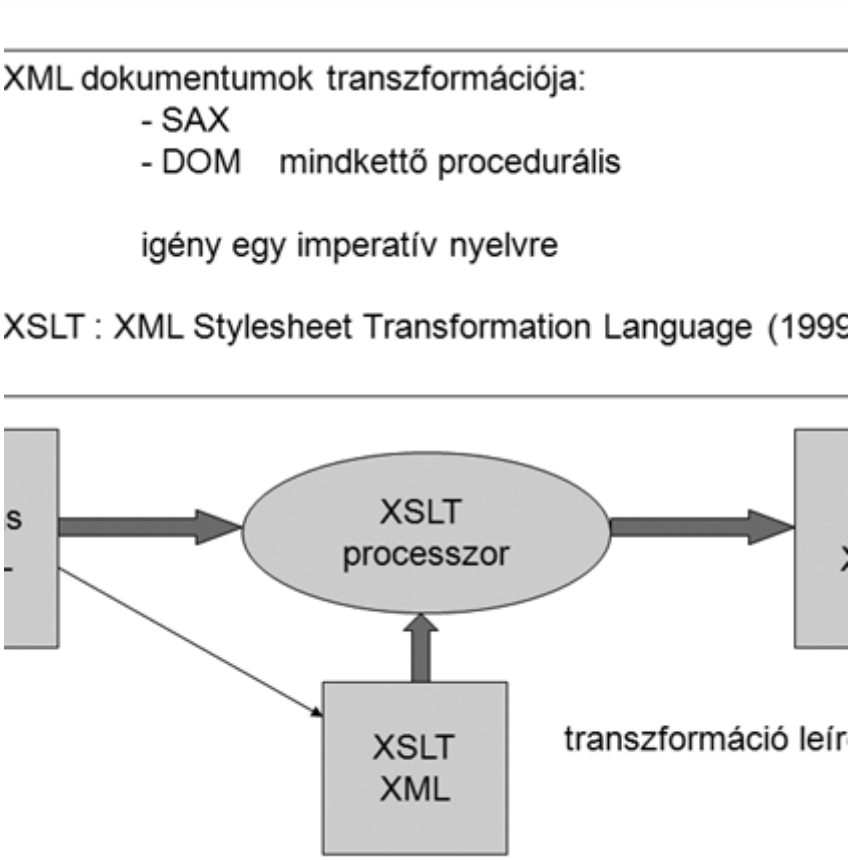
- XSLT értelmező.

A legfontosabb dolog egy jól működő XSLT értelmező beszerzése. A gyakorlatok során most is támaszkodhatunk az eddig is használt Oxygen XML szerkesztőre. Ezen termék fő előnye, hogy az XSLT program megírásánál nagy támogatást kapunk a rendelkezésre álló parancsok és a nyelvhelyességet illetően. Az Oxygen szerkesztő esetében az alábbi lépéseken keresztül tudjuk kipróbálni az XSLT működését:

1. forrás állomány létrehozása: File menüpont New (XML) alpontján keresztül,

2. transzformáció leíró állomány létrehozása: File menüpont New (XSL Stylesheet) alpontján keresztül,

3. transzformáció végrehajtása: Document menüpont Transformation alpontján keresztül.



2. Az XSLT működési sémája

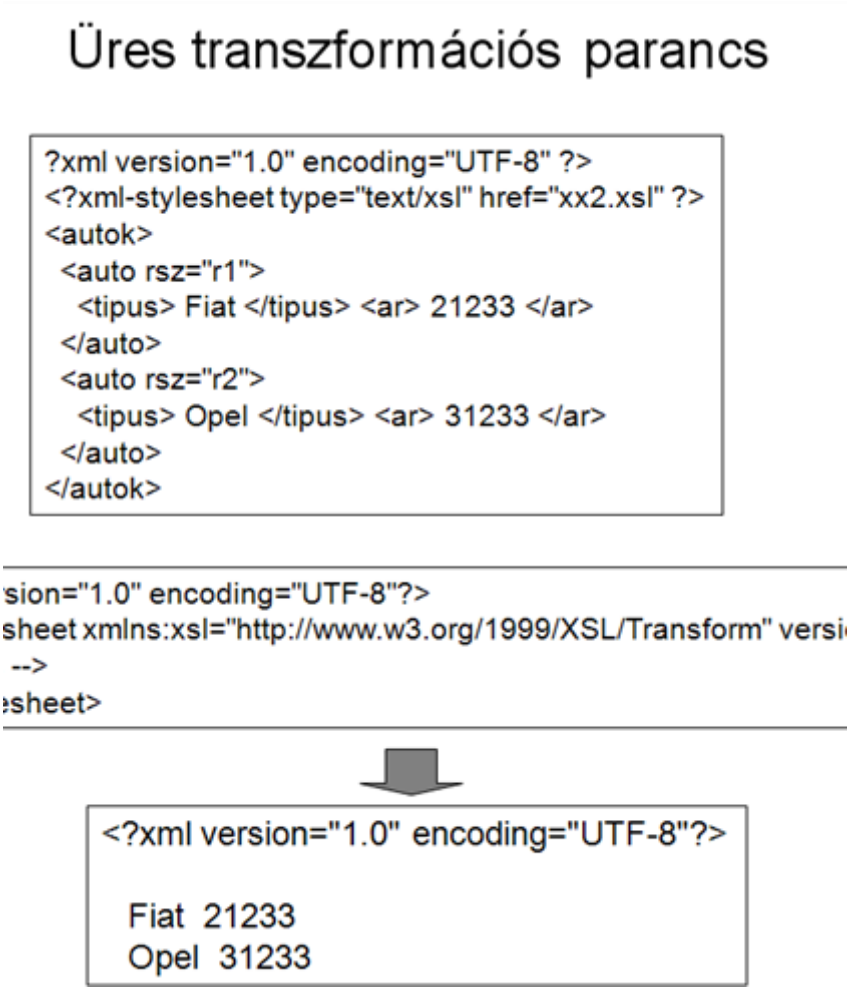
Szerencsére, a világhálón több ingyenes XSLT értelmező is elérhető. Talán a legtöbbet használt termék ezek közül az Eclipse IDE keretrendszer és a MS Internet Explorer böngészője. Az Eclipse komplett XML adat és séma szerkesztővel bír, míg a böngésző beépített XSLT értelmezővel és séma ellenőrzővel rendelkezik, melyekkel az elkészült XML állományok kipróbálhatók.

Az XSLT forráskódot XML állományként tároljuk. A forrásállomány gyökér eleme egy

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
  <!-- transzformációs minták -->
</xsl:stylesheet>
```

elem. Az XSLT transzformációs parancsok mindegyike az itt definiált 'http://www.w3.org/1999/XSL/Transform' névtérhez tartoznak. A transzformációs parancsok kiterjednek az előzőekben említett mintaillesztés mellett többek között az alábbi műveletekre:

- csomópontok rendezése a feldolgozáshoz,
- kifejezések megadása az eredmény XML dokumentum előállításához,
- csomópontok feldolgozási ciklusa,
- változók létrehozása,
- az eredmény dokumentumba új struktúra elem létrehozatala,
- feltételes transzformáció végrehajtása.



3. Az üres transzformációs parancssor működése

A forrás XML állományban jelöljük ki, hogy az adott állományon egy transzformációt kell végrehajtani. A transzformációs állomány kijelölése egy

```
<?xml-stylesheet type="text/xsl" href="forrás-xsl" ?>
```

elemmel történik. Ez a feldolgozási direktíva a dokumentum gyökéreleme előtt szerepel. A 'href' elemjellemzőnél specifikáljuk a transzformációt leíró XSLT állományt. A következő kis minta bemutat egy

X1.XML forrás dokumentumot, egy CS1.XSL transzformációt megadó állományt és a transzformáció eredményét tartalmazó X2.XML állományt.

X1.XML:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="xs1.xsl" ?>

<autok>
  <auto rsz ="r11"> <tipus> Fiat </tipus> <ar>21233</ar> </auto>

  <auto rsz ="r21"> <tipus> Opel </tipus> <ar> 41233</ar> </auto>

  <auto rsz ="r31"> <tipus> Honda </tipus> <ar>71233</ar> </auto>

</autok>
```

XS1.XSL:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="auto">
  <xsl:value-of select ="./@rsz"/>:   <b> <xsl:apply-templates/> </b> <br/>
</xsl:template>

<xsl:template match="tipus">
  <i> <xsl:apply-templates/> </i>
</xsl:template>

</xsl:stylesheet>
```

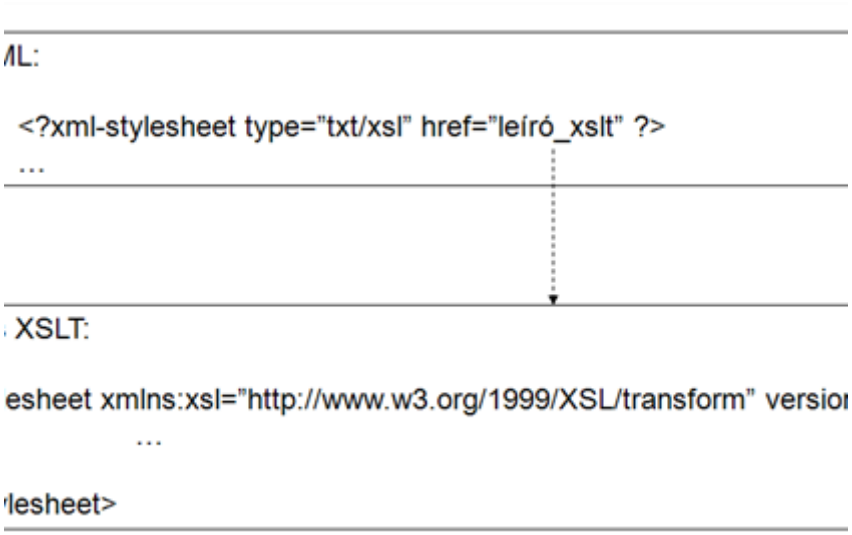
X2.XML:

```
r11: <b> <i> Fiat </i> 21233 </b> <br/>

r21: <b> <i> Opel </i> 41233 </b> <br/>

r31: <b> <i> Honda </i> 71233 </b> <br/>
```

A transzformációt leíró kódban a 'match' és 'select' elemjellemzők értékeként egy-egy XPath kifejezést adunk meg, melyek kijelölik, hogy mely csomópontoknál kell használni a kapcsolódó transzformációt, illetve mely csomópont értékét kell kiíratni az eredmény dokumentumba. A későbbiekben látni fogjuk, hogy a példában szereplő egyszerűbb kifejezések mellett jóval összetettebb XPath formulák is képezhetők.



4. A forrás és séma file összekapcsolása

A következő animációban bemutatjuk, hogy az Oxygen XML szerkesztőben milyen lépéseken keresztül lehet XSLT transzformációs állományt létrehozni, illetve hogyan lehet az elkészült parancssort végrehajtani egy forrás XML állományon: **Oxygen demo animáció**.

2. Az XSLT működési elve

Az XSLT feldolgozók bemenete a feldolgozandó XML dokumentumból készített dokumentumfa. Az XSLT transzformációs műveletek mindegyike a fára vonatkozik. A feldolgozás kimenete az eredményfa linearizálásával kapott XML dokumentum lesz. A dokumentumfa feldolgozásának elve a fa csomópontjainak bejárásán alapszik, azaz a feldolgozó a gyökér csomóponttól kezdve bejárja a teljes fát, s amely csomóponthoz tartozik átalakítási, eredmény generálási szabály, ott előállítja a kért XML dokumentum részletet. Az eredményfa ezen részek összevonásával áll elő. Alapesetben a feldolgozás a gyökértől halad a gyerekek felé, egy top-down, deep-first mélységi bejárás megközelítésben. Az alapértelmezés szerinti feldolgozás pontos menete:

1. A feldolgozó rááll a gyökér csomópontra.

2. Megnézi, hogy létezik-e feldolgozási utasítás erre a csomópontra. Ha nem létezik ilyen minta és vannak gyerek elemek, akkor sorra veszi a gyerek elemeket, és mindegyikre rekurzívan elvégzi ezt a vizsgálatot. Ha nincs gyerek, megáll a feldolgozási lánc. Előtte, ha szöveg csomópontnál tart a feldolgozás, akkor kiírja a szövegelemek tartalmát.

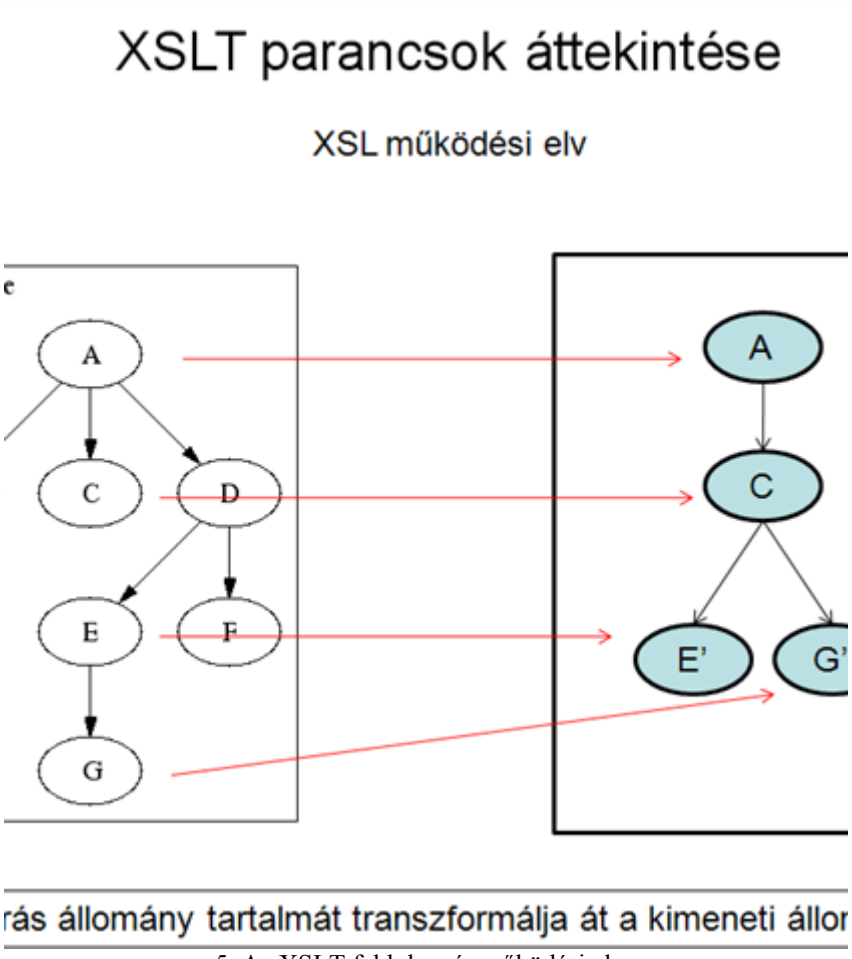
3. Ha van illeszkedő parancs minta, akkor azt kiértékelve előáll egy eredményfa részlet. Alapesetben a csomópont feldolgozása után megáll a feldolgozási lánc, a gyerekek nem kerülnek ellenőrzésre. Lehet azonban továbbhaladási igényt is adni, ekkor a kijelölt elemekre megy tovább a feldolgozás. A továbblépés tetszőleges, XPath formátumban megadható célhelyekre történhet.

Természetesen ez a leírás csak egy leegyszerűsített algoritmust vázol fel, hiszen a tényleges feldolgozás során több speciális problémát is meg kellett oldani. Egyik ilyen feladat, hogy a feldolgozó tartalmazzon-e alapértelmezett transzformációs szabályokat vagy sem. Ha nincs ilyen szabály, akkor a feldolgozó leáll a mélységi bejárással minden olyan csomópontnál, amelyhez nem adtunk meg feldolgozási utasítást. Ha viszont van alapértelmezett feldolgozó parancs, akkor minden ilyen esetben ez a beépített parancs kerül végrehajtásra, s továbbfuthat a mélységi bejárás. Az XSLT működési szabályban az az eset valósul meg, amikor értelmezett egy alapértelmezett feldolgozó utasítás. Emiatt, ha a felhasználó egyetlen egyedi transzformációt sem definiál, akkor is lesz eredmény XML fa. Az alapértelmezett feldolgozási szabályok:

- minden elemnél (nem beleértve az elemjellemzőt és névteret), ha az nem szövegcsomópont, az eredményfa bővítése nélkül továbblép a gyerek elemek (nem beleértve a jellemzőket és névteret) felé,

- a szövegcsomópont esetén kiírja az eredménybe ezt a csomópontot (a szövegtartalmat),

- egyéb csomópontokat figyelmen kívül hagy.



5. Az XSLT feldolgozás működési elve

Vegyünk például egy alap XML dokumentumot:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="xx2.xsl" ?>

<autok>
  <auto rsz="r1">
    <tipus> Fiat </tipus> <ar> 21233 </ar>
  </auto>
  <auto rsz="r2">
    <tipus> Opel </tipus> <ar> 31233 </ar>
  </auto>
</autok>
```

és egy alap, üres transzformáció leírót:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

```
  <!-- üres -->
</xsl:stylesheet>
```

Az eredmény XML dokumentum alakja:

```
<?xml version="1.0" encoding="UTF-8"?>

 Fiat  21233
 Opel  31233
```

Mint látható, a kapott XML dokumentum nem feltétlenül illeszkedik a helyesen formáltság követelményeire, hiszen a példában nincs gyökér eleme a dokumentumnak, a szövegrészek elemen kívül helyezkednek el. Tehát az XSLT transzformáció nem garantálja helyesen formáltság feltételeit az eredmény dokumentumok esetében. Ezért a fejlesztőknek kell ügyelniük, hogy olyan transzformációkat hívjanak meg, melyek együttesen biztosítják a helyesen formáltság kritériumait. A feldolgozás másik speciális problémája a többértelműség, vagyis amikor egyazon csomóponthoz több szabály is definiálva van. Természetesen az alapértelmezési szabály csak akkor kerül felhasználásra, ha nincs explicit szabály megadva. De az explicit szabályok között is lehet több olyan, amelyek mindegyike érvényes egyazon csomópontnál. A konfliktus kezelés egyik legegyszerűbb megvalósítása, amikor egy definíciós sorrendiséget vesznek alapul. A tapasztalatok szerint a legkésőbb definiált szabálynak lesz a legnagyobb a prioritása. Emiatt a transzformáció eredménye nemcsak a megadott szabályok halmazától, hanem azok definíciós sorrendjétől is függ.

Az XSLT feldolgozását végző motor általános struktúráját a az egyik elterjedt XSLT feldolgozót, a Saxon rendszer példája alapján ismertjük. A motor bemeneti oldalán találhatók a fa felépítő egység (Tree Constructor). Ez az egység mind az XSLT parancsállományt, mind a feldolgozandó XML állományt átalakítja egy DOM-hoz hasonló XML fává. Az egység három komponenst tartalmaz. Az első komponens a SAX API-ra épülve lineárisan átolvassa a forrásként megadott XML, XSLT dokumentumokat. A második modul a generált köztes dokumentum leíráson egy normalizálást hajt végre, azaz eltávolítja a felesleges szóköz jellegű karaktereket. A SAX által generált eseményekre épülve a harmadik modul építi fel a csomópont objektumok fáját. A XSLT programfa ezt követően egy XSLT fordítóhoz (XSLT Compiler) kerül. E modul feladata a fa elemzése és a fa konvertálása egy tömörített utasításkódra. Az ellenőrzés kiterjed többek között a parancsok szintaktikájára, az XPath kifejezésekre és a kereszthivatkozásokra. A modul az XPath elemeket külön kiemeli, s átalakítja egy belső formátumra. Az eredményül kapott, ellenőrzött és elő feldolgozott fát nevezik jelzett XSL fának (decorated XSLT tree). Az elő feldolgozás fő célja a tényleges transzformáció végrehajtás optimalizálása. Az optimalizálás fő eszköze egy döntési fa, melynek szerepe az XML csomópontokhoz tartozó transzformációs minták gyors megkeresésében van. Vagyis a tényleges transzformáció során a feldolgozó nem lineáris kereséssel nézi át a lehetséges mintákat, hanem egy előre felépített döntési fával dolgozik. A döntési fa mellett egyéb optimalizálási lépések is felhasználásra kerülnek, mint például a

- szelekciós feltételek egyszerűsítése,
- jellemzők közvetlen elérésének megvalósítása,
- változók elérésének gyorsítása.

A rendszer következő egysége a navigációs modul (Tree Navigator), melynek célja a fa mélységi bejárása. A gyökér elemtől kiindulva a gyerekek felé halad a bejárás. A feldolgozott, kontextus csomóponthoz rendszer megkeresi az illeszkedő mintát az előbb említett döntési fa segítségével. Több illeszkedő minta esetén egy konfliktuskezelő algoritmus dönti el, hogy melyik minta legyen a végrehajtandó minta. A mintákhoz kapcsolódik egy XSLT transzformáció leíró utasítássorozat. A transzformáció tényleges végrehajtását az XSLT értelmező modul (XSLT Interpreter) végzi el. Az XSLT fordító egy tömörített kódot állít elő, melyhez ezen értelmező társítja a ténylegesen futtatandó programkódot. A programkód célja felépíteni egy eredmény XML

fát, melyet majd a végén egy XML dokumentumba lehet lementeni. A Saxon implementációban minden XSLT parancshoz egy saját Java osztály tartozik. Az XML dokumentum előállítását a kiíró modul (Outputter) végzi el.

Az XSLT nyelvben több transzformációs parancs is létezik, melyekből a legfontosabbnak a már többször említett mintaillesztési és navigációs utasítások tekinthetők. A további lehetőségeket itt most csak röviden felsorolva:

- feltételes transzformáció
- változókezelés
- névvel azonosított transzformációk
- XML elemek létrehozása
- elemek rendezése
- csoportképzés
- aggregáció
- segédfüggvények
- XSLT állományok összefűzése

A következő animáció egy példán keresztül bemutatja az XSLT transzformációs mechanizmusát: **XSL működési mechanizmus demo animáció**. A XSLT nyelv parancsainak részletes leírását a következő fejezet adja meg.

3. Az XSLT parancsai

3.1. Mintaillesztés

A dokumentumfa egy megadott csomóponthalmazának kijelölése a

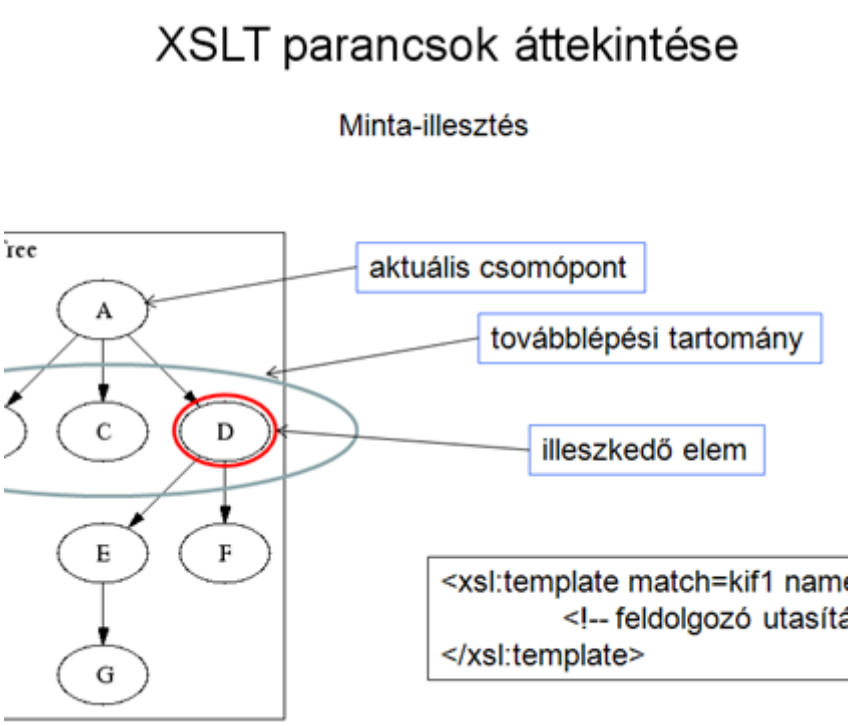
```
<xsl:template match=kif1 name=kif2 >
   <!-- feldolgozó utasítások -->
</xsl:template>
```

paranccsal történik. A tagban szereplő 'kif1' jellemző egy illesztési mintát jelöl, mely a feldolgozandó csomópontokat jelöli ki a nevük alapján. A 'kif2' egy nevet hordoz, megadásával egy azonosító név köthető ezen mintához. Mint korábban már említettük, alapesetben a minta feldolgozása után a fa mélységi bejárása megáll. Vegyük például a

```
<xsl:template match="/"  >
   UDV!
</xsl:template>
```

parancsállományt, amely a gyökér elemre illő mintát tartalmaz, melyben egy szöveg konstans a generált kimenet. A feldolgozó a szöveg kiírása után leáll a fa feldolgozásával. Az így kapott eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>
 UDV!
```



6. Minta illesztés XSLT parancsa

A minta egynél több csomópontot is tartalmazhat, ekkor mindegyiknél végrehajtódik az eredménygenerálás. Vegyük az alábbi példát:

```
<xsl:template match="auto"  >
 udv!
</xsl:template>
```

A példában az XML dokumentum auto nevű elemeinél lesz illeszkedés. Az XSLT 'template' mintaillesztési mechanizmusa nem teljesen egyezik meg az XPath szabvánnyal. A mintában használt kifejezések ugyan mind XPath kifejezések is egyben, de a fordított irányban ez már nem állítható, vagyis nem minden XPath kifejezést lehet felhasználni mintaként. A legfontosabb megkötések:

- ez elemi mintákat a '|' operátorral lehet összekötni, amely megfelel a halmaz uniónak
- az elemi elérési lépésekben csak kétféle tengely szerepelhet: a 'child' és az 'attribute' tengely. Más navigációs irány nem megengedett. A csomópont kiválasztó és szelekciós rész korlátozás nélkül használható.
- létezik egy speciális minta, az

```
id(kifejezés)
```

amely azon csomópontokat adja vissza, ahol az ID tulajdonságú jellemző értéke megegyezik a zárójelben megadott kifejezéssel.

- a child:: tengelyirányokat egyes esetekben úgy módosítja, hogy maga a kontextus csomópont is belefoglalt legyen az irányba.

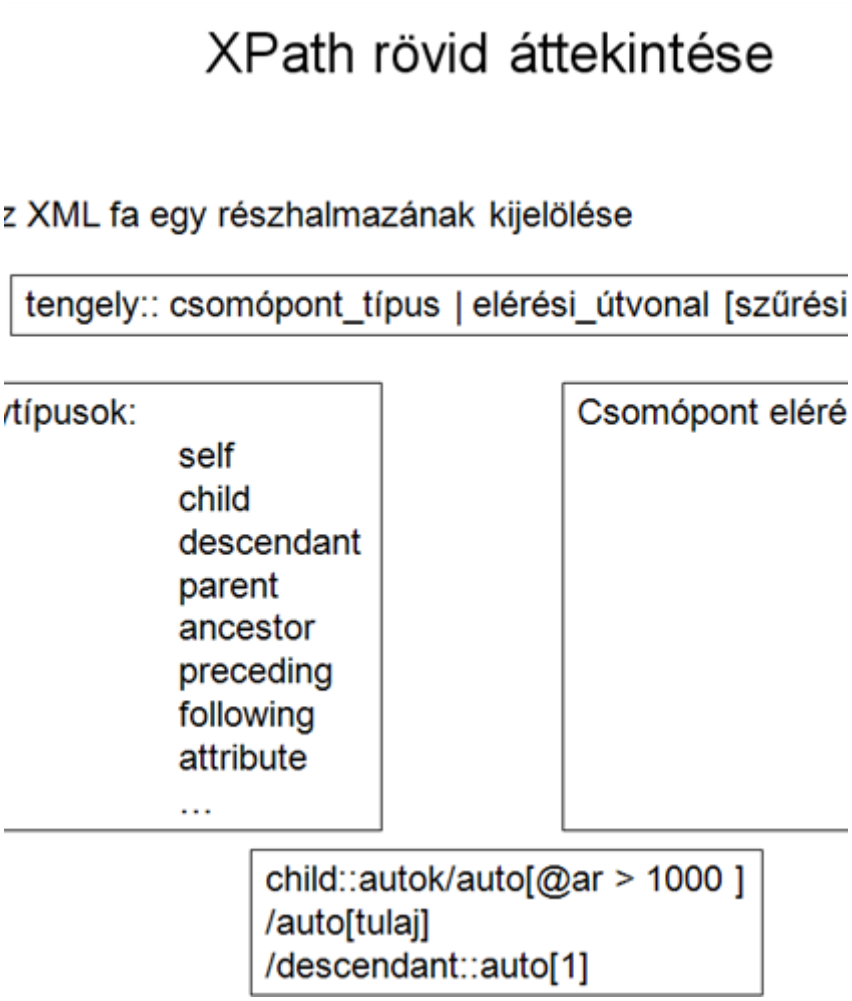
Az XSLT minta kiértékelésének pontos ismerete is igen fontos a megfelelő XSLT állomány elkészítésénél. A kiértékelés algoritmusa a következőkben foglalható össze. A kiértékelő motor elsőként elmegy a gyökér (dokumentum) elemhez. Megnézi, van-e az XSLT parancsfában illeszkedő minta a gyökérre. Ha nincs, akkor az alapértelmezési parancsot hajtja végre, ami annyit mond ki, hogy menjen tovább a dokumentumfa mélységi bejárása a gyerek csomópontok felé (az eleme jellemző és névtér nem tartozik ide). A szöveges csomópontok esetében az alapértelmezési szabály a szöveges tartalom kiírását írja elő. Mivel ez levél csomópont, a mélységi keresés is leáll. Explicit feldolgozási parancs esetén, ha az nem tartalmaz továbblépést, akkor megáll a fa mélységi bejárása az adott csomópontnál. A feldolgozó motor az illeszkedés a minták vizsgálat során megkeresi, hogy mely minta illeszkedik a fa feldolgozás alatt álló csomópontjára. Egy p minta akkor illeszkedik az x csomópontra, ha a fában létezik olyan y csomópont, melyre, mint kontexus elemre kiértékelve a p mintát, az eredményhalmaz magába foglalja az x-et. Ez formálisan úgy is leírható, hogy egy (x,p) páros viszonyában a feldolgozó a

```
root(.)//(p)
```

XPath kifejezést értékeli ki, s ennek eredményhalmazában kell szerepelnie a x csomópontnak.

Gyakorlásként néhány tipikus mintát mutatunk be:

- nev : a fa összes olyan eleme, melynek azonosító neve 'nev', ekvivalens alakja root(.)//nev,
- / : a dokumentum gyökér csomópontja,
- /nev : a fa gyökér eleme, amely a dokumentum gyökér csomópont alatt helyezkedik el,
- nev[2] : azon elemek, melyek a szüleik második 'nev' nevű gyerekei,
- text() : a dokumentum összes szövegtípusú eleme,
- * : a dokumentum összes névvel rendelkező eleme (ebben a halmazban a szövegcsomópontok, jellemzők, névterek nem szerepelnek),
- node() : a dokumentum-fa összes elem csomópontja (elemjellemző, névtér nem szerepel benne),
- attribute::nev : a fa összes 'nev' azonosítójú elemjellemzője,
- nev1 | nev2[@nev3=kif3] : az eredmény két csomóponthalmaz uniója. Az egyik részhalmazba a 'nev1' névvel rendelkező elemek, míg a másikba azon 'nev2'-vel rendelkező elemek kerülnek, melyekhez tartozik egy 'nev3' nevű jellemző, és annak értéke 'kif3' .



7. XPath szabvány szintakszisának áttekintése

Ahhoz, hogy eredményhalmazban lévő csomópontot feldolgozzon az XSLT motor, a kapcsolódó minta mellett még arra is szükség van, hogy a fa mélységi bejárás elérje a csomópontot. Vegyük például az alábbi mintát:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
 <xsl:template match="/">
  Hello
 </xsl:template>
 <xsl:template match="auto">
  Auto
 </xsl:template>
</xsl:stylesheet>
```

XSLT állományban hiába szerepel az 'auto' nevű elemek feldolgozása, a feldolgozó nem fogja ezt elvégezni, mivel a mélységi bejárás nem jut le a 'auto' nevű csomópontokig. Az elakadás oka, hogy a dokumentum gyökér csomóponthoz létezik explicit minta, s ez fog elsőként végrehajtódni. Ezen parancsrészben nem szerepel továbblépési utasítás, csak egy 'Hello' szöveg kiíratása, tehát megáll a fa mélységi bejárása. A fa bejárás folytatásához egy újabb parancsra az 'apply-template' parancsra van szükség.

3.2. Feldolgozás továbbléptetése más csomópontokra

A fa-bejárás folytatásának parancsalakja:

```
<xsl:apply-templates select = "kif1" >
  <!-- rendezés -->
</xsl:apply-templates>
```

Az apply-template utasítás a keresést továbbfolytatja a 'select' jellemzőn keresztül kijelölt csomóponthalmazban. A parancs 'select' jellemzője opcionális tag. Ha nem szerepel, a keresés a csomópont összes gyerekeleménél folytatódik, azaz a 'select' jellemző alapértelmezési értéke:

```
child::node()
```

A 'select'-nél megadott csomópont kiválasztási feltétel is az XPath szabványon alapszik. Az itt alkalmazható formulák köre tágabb, mint amit a 'match' opció megengedett: tetszőleges tengely irányokba lehet mozogni. A következő példákban az előbb bemutatott XSLT állományt módosítjuk úgy, hogy a feldolgozás továbbmenjen a gyökér összes 'auto' nevű leszármazottja felé:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
  <xsl:template match="/">
     Hello
     <xsl:apply-templates select="descendant::auto"/>
   </xsl:template>
   <xsl:template match="auto">
      Auto
   </xsl:template>
</xsl:stylesheet>
```

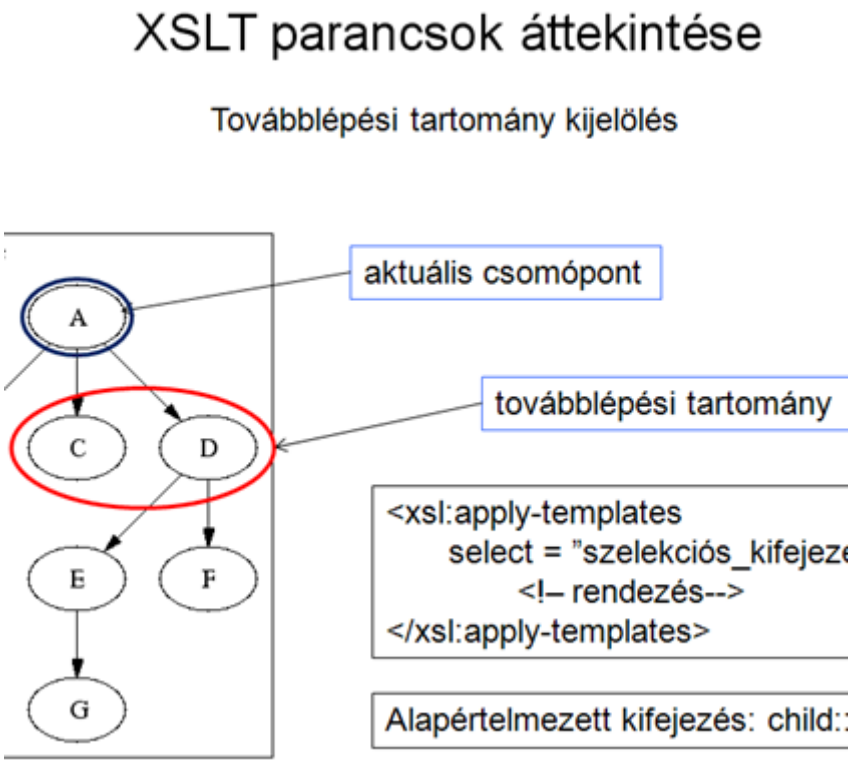
Ekkor elsőként a gyökérelemet dolgozza fel, melynek során végrehajtja a továbblépési utasítást is. A feldolgozott továbblépési elemek a gyökér elem 'auto' nevű leszármazottai lesznek. Azáltal, hogy a 'select' tulajdonság nemcsak mélységi keresést tesz lehetővé, a fa tetszőleges irányokban haladva járható be. A nagy szabadság viszont végrehajtási problémákhoz is vezethet, hiszen kialakulhat többek között végtelen ciklus is. Vegyük az alábbi, formálisan helyes XSLT parancssort:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
  <xsl:template match="autok">
    p:
     <xsl:apply-templates select="child::auto"/>
   </xsl:template>
   <xsl:template match="auto">
    c:
     <xsl:apply-templates select="parent::node()"/>
  </xsl:template>
</xsl:stylesheet>
```

ahol az adatállomány:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="x1.xsl" ?>
<autok>
  <auto rsz="r1">
    <tipus> Fiat </tipus> <ar> 21233 </ar>
  </auto>
  <auto rsz="r2">
    <tipus> Opel </tipus> <ar> 31233 </ar>
  </auto>
</autok>
```

A fenti XSLT transzformációt végrehajtva egy futási hibát ('Too many nested apply-templates calls') kapunk, amely arra utal, hogy túl sok egymásba ágyazott hívás jött létre. Ugyanis az első minta végrehajtása során továbblép a gyerek 'auto' elemre. Ott a második minta aktivizálódik, s továbblép a szülő ('autok') elemre, ekkor viszont újra az első minta lesz aktív, záródik a kör, s egy végtelen végrehajtási ciklus alakult ki.



8. Feldolgozás továbblépés XSLT parancsa

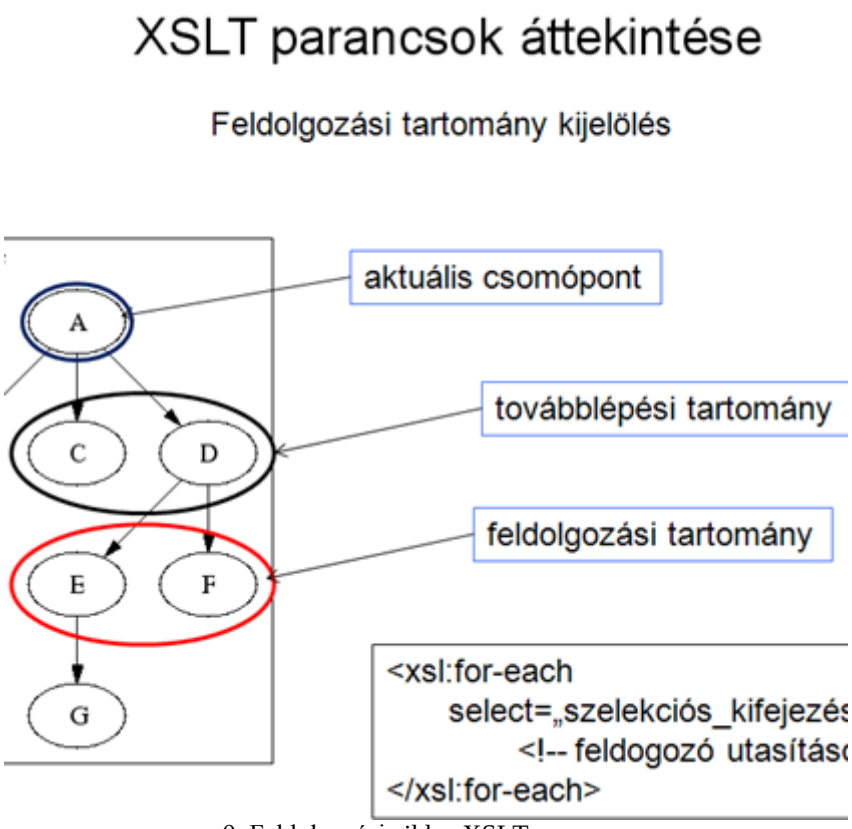
Az apply-template utasítás meghatározza a következő lépésekben feldolgozza a csomópontok halmazát, de a tényleges művelet végrehajtást a többi 'template' utasításokra hagyja. Van ezen mechanizmus mellett egy olyan lehetőség is, amikor egy egységben adjuk meg, hogy mely csomópontokon milyen műveleteket kell végrehajtani. Ez a feldolgozási ciklus utasítása.

3.3. Ciklus utasítás

A feldolgozó ciklusnál paraméterként egy csomópont szelekciós kifejezés szerepel, melyet XPath formátumban kell megadni. Az utasítás hatására az eredményhalmaz minden egyes elemén végrehajtódik az utasítás magjában megadott műveletsor. Az utasítás szintaktikája:

```
<xsl:for-each  select="kif1">
  <!-- feldogozó utasítások -->
</xsl:for-each>
```

Az itt szereplő 'select' rész jelöli ki a feldolgozandó csomópont halmazt. A feldolgozó utasítások rendszerint az eredményfa tartalmát határozzák meg. Az előző példákban mindig szöveg konstansokat írattunk ki az eredménytáblába, de legtöbbször a forrásállományból származtatott, attól függő tartalomra van szükség. A forrás fa elemeinek áthozatalára egy önálló utasítás szolgál.



9. Feldolgozási ciklus XSLT parancsa

A ciklus esetében fontos lehet az elemek elérési sorrendje is. Ez a rendezésre szolgáló utasítással valósítható meg, amely egy csomóponthalmazhoz köthető, formátuma:

```
<xsl:sort select="kif1"  order="kif2"   collation="kif3" />
```

Az első kifejezés megadja, hogy mely csomópont tárolja a rendezés kulcsát. A 'kif1' egy XPath szabvány szerinti kifejezést takar. Az 'order' tulajdonság a rendezés irányát (csökkenő vagy növekvő) állítja be. A harmadik, kif3 tulajdonság a rendezési sorrendet határozza meg. Ez adja meg az egyes karakterek közötti megelőzési relációt. Példaként alakítsuk át az előző példát úgy, hogy a típusnevek ABC sorrendben jelenjenek meg:

```
<?xml version="1.0" ?>
 <xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
   <xsl:template match="autok">
     <xsl:for-each select="auto">
       <xsl:sort select="tipus"/>
```

```
      <xsl:value-of select="tipus"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

3.4. Kifejezések kiértékelése

A 'value-of' utasítás segítségével lehet a forrásállomány alapján képzett kifejezéseket kitenni a célállományba. A parancs alakja:

```
<xsl:value-of select="kif1" />
```

A 'kif1' jelöli ki a kiíratandó értékeket. A kifejezés jelölhet csomópontot is, ebben az esetben a csomópont szöveges tartalma, és nem maga a csomópont kerül bemásolásra az eredményfába. A kifejezés lehet azonosító XPath útvonal mellett numerikus és szöveges érték is. Példaként vegyük az alábbi transzformáció leírást:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
  <xsl:template match="autok">
    <xsl:for-each select="auto">
      <xsl:value-of select="tipus"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Eredményül az 'autok' elem 'auto' gyerekeinek 'tipus' gyerekének szövegértékeit kapjuk meg, hiszen a 'value-of' utasítás 'select' paramétere most egy azonosító nevet tartalmaz, tehát ez egy xPath csomópont kijelölő kifejezésnek tekinthető. Ekkor a csomópont mögötti szöveges érték kerül át. A lista dokumentum sorrendben adja vissza a szöveges csomópontok értékeit. Ha az előző mintában a 'value-of' utasítást a

```
<xsl:value-of select="'tipus'"/>
```

alakban adjuk meg, az eredményben a 'típus' szöveg konstans jelenik meg annyiszor, ahány 'auto' gyerekeleme van az 'autok' csomópontnak. A 'select' tagban megadott kifejezést az értelmező kiértékeli, s a kapott eredményt teszi ki kimenetre. Ha például egy numerikus kifejezést, pl. 1+2-t adjuk meg, akkor az eredményben a 3-as érték jelenik meg, azaz a

```
t=<xsl:value-of select="1+2"/>
```

esetében az eredmény az alábbi alakú lesz:

```
t=3
```

A 'value-of' utasítás csak elemi szöveges értékeket másol át az eredménybe a forrás fából. Ha egy teljes farészletet, elemhierarchiát szeretnénk átvinni, akkor annak legegyszerűbb módja a csomópont másolási utasítások használata. A csomópontok átmásolásához kétféle utasítás áll rendelkezésre. Az elemi üres csomópontot viszi át a 'copy' parancs, míg a párja a 'copy-of' teljes csomópont részfa átmásolását elvégzi. Az egyszerűbb, aktuális csomópontot átvivő utasítás alakja:

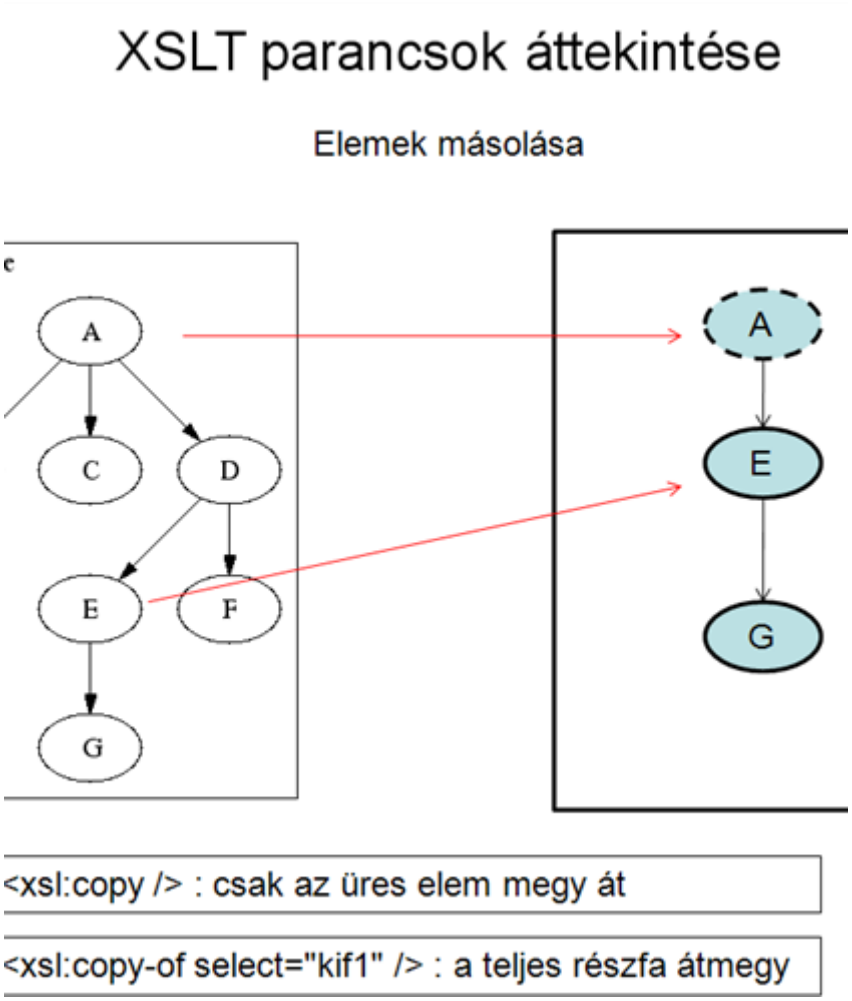
```
<xsl:copy />
```

A teljes részfát átvivő utasításnál a formátum:

```
<xsl:copy-of select="kif1" />
```

A 'kif1' kifejezés XPath formátumban megadja, hogy mely csomópont alatti részt kell átmásolni. Ha kifejezés több csomópontot is kijelöl, akkor mindegyik alatti részfa kiírásra kerül. A teljes dokumentum átmásolásának legegyszerűbb módja a következő parancssor:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"' version="1.0">
  <xsl:template match="/">
    <xsl:copy-of select="." />
  </xsl:template>
</xsl:stylesheet>
```



10. Elemek másolásának XSLT parancsa

A forrás dokumentum átalakítása során nemcsak csomópont átmásolás és törlés igénye léphet fel, hanem új csomópontok létrehozásának szükségessége is. Az XSLT több kapcsolódó utasítást tartalmaz a különböző csomópont típusok megalkotására.

3.5. Csomópontok létrehozása

Új elemek elhelyezéséhez a csomópontok létrehozására szolgáló

```
<xsl:element name="nev" >
  <!-- tartalom -->
</xsl:element>
```

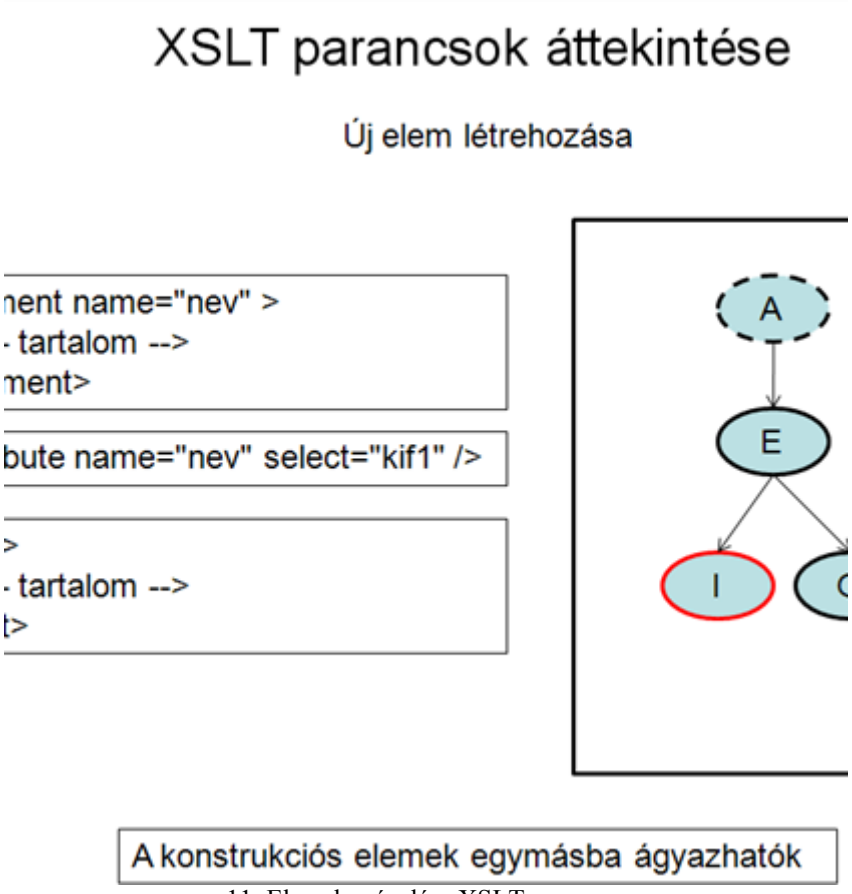
utasítás használható fel. Ha egy új elemjellemzőre van szükség, akkor a

```
<xsl:attribute name="nev" select="kif1" />
```

utasítást használhatjuk, melyben a select tulajdonság a létrehozott elemjellemző értékét adja meg. A fenti két csomópont típus mellett további csomópont létrehozó utasítások léteznek, minden egyedi csomópont típusnak megvan a saját létrehozó utasítása. Így például szövegcsomópontot a

```
<xsl:text>
  <!-- tartalom -->
</xsl:text>
```

alakban a hozhatunk létre.



11. Elemek másolása XSLT parancsa

3.6. Feltételes utasítás végrehajtás

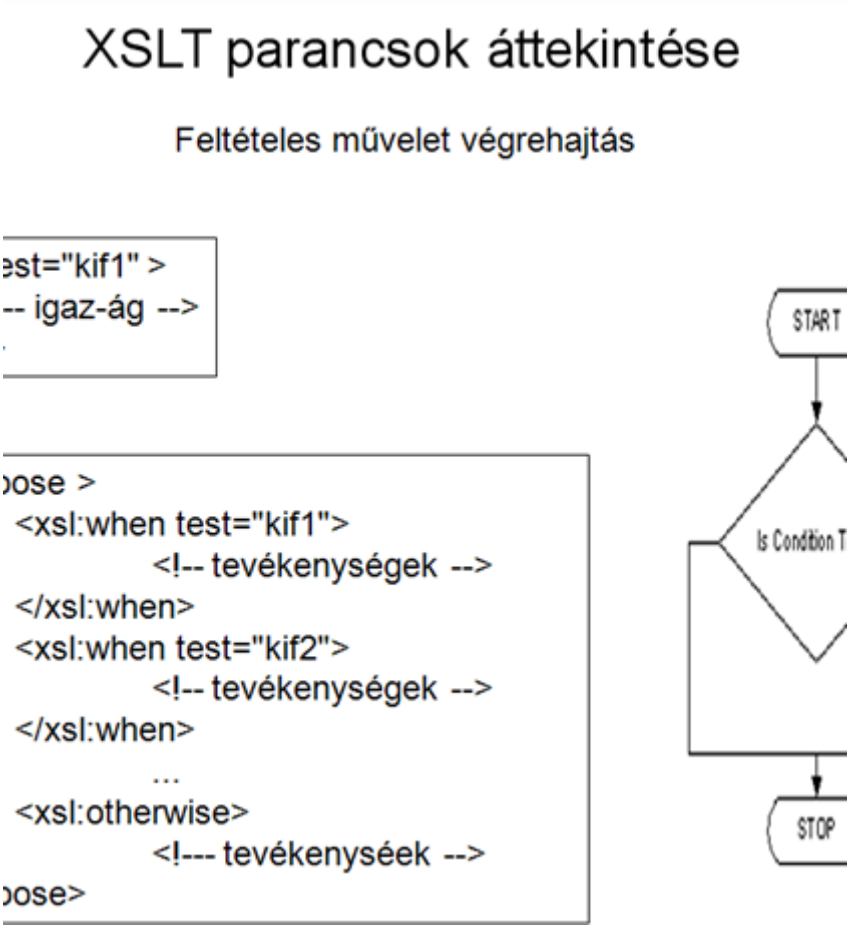
Mivel a létrehozott eredmény dokumentum tartalma több különböző, futási időben meghatározott paramétertől is függhet, szükség van olyan utasításra, amely feltételhez kötött végrehajtást tesz lehetővé. Ezen feltétel végrehajtási részt is kétféle módon jelölhetjük. Az egyszerűbb esetben csak egy igazág létezik, s ekkor a :

```
<xsl:if test="kif1" >
  <!-- igaz-ág -->
</xsl:if>
```

utasítás csak akkor hajtja végre az igazágban megadott utasításokat, ha 'kif1' kifejezés is igaz értéket ad vissza. A kifejezésnek logikai értékűnek kell lennie. Ha több különböző tevékenység közül választunk egyet, akkor a többszörös elágazás utasítását használjuk. Az utasítás alakja:

```
<xsl:choose >
  <xsl:when test="kif1">
     <!-- tevékenységek -->
  </xsl:when>
   <xsl:when test="kif2">
     <!-- tevékenységek -->
  </xsl:when>
   ...
   <xsl:otherwise>
       <!--- tevékenységek -->
   </xsl:otherwise>
</xsl:choose>
```

A szerkezetben mindegyikághoz egy logikai kifejezés tartozik. A vezérlés a sorrendben első, igaz értékű kifejezéshez tartozó ágat választja ki. Ha egyetlen egy ilyen 'when' ág sincs, akkor az 'otherwise' ágra kerül a vezérlés.



12. Feltételes végrehajtás XSLT parancsa

Példaként vegyük azt az esetet, melyben a vizsgaleíró dokumentumban a számmal megadott vizsgajegyet szöveges leírással is cseréljük le. Az induló forrásdokumentum:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="x1.xsl" ?>
<vizsgak>
 <vizsga>
     <targy> Metamatika </targy> <diak> K234 </diak> <jegy> 2 </jegy>
 </vizsga>
 <vizsga>
   <targy> Metamatika </targy> <diak> K761 </diak> <jegy> 3 </jegy>
 </vizsga>
</vizsgak>
```

A kapcsolódó XSLT leírás:

```
 <?xml version="1.0" ?>
 <xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"'
 version="1.0">
  <xsl:template match="/">
    <xsl:copy>
      <xsl:apply-templates/>
     </xsl:copy>
  </xsl:template>

  <xsl:template match="vizsgak">
    <xsl:copy>
      <xsl:apply-templates/>
     </xsl:copy>
  </xsl:template>

  <xsl:template match="vizsga">
    <xsl:copy>
      <xsl:for-each select="*" >
        <xsl:choose>
<xsl:when test="name()='jegy'">
   <xsl:copy>
     <xsl:choose>
       <xsl:when test=" text() = 1 ">
          <xsl:text> egy </xsl:text>
       </xsl:when>
       <xsl:when test=" text() = 2 ">
           <xsl:text> ketto </xsl:text>
       </xsl:when>
       <xsl:when test=" text() = 3 ">
          <xsl:text> harom </xsl:text>
       </xsl:when>
       <xsl:when test=" text() = 4 ">
          <xsl:text> negy </xsl:text>
       </xsl:when>
       <xsl:when test=" text() = 5 ">
          <xsl:text> ot </xsl:text>
       </xsl:when>
     </xsl:choose>
              </xsl:copy>
  </xsl:when>
            <xsl:otherwise>
```

```
      <xsl:copy-of select="." />
    </xsl:otherwise>
 </xsl:choose>
        </xsl:for-each>
     </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

A három minta közül az első a gyökér csomópontot másolja át, gyerekeit a további minták határozzák meg. A második minta a gyökérelemet ('vizsgak') teszi le az eredménybe. A harmadik minta végzi el a 'vizsga' elem átvitelét. Ehhez ciklusban a gyerekek részfáját veszi sorra, s a típusoktól függően átmásol (otherwise) vagy átalakít ('jegy' ág). Ezen esetben csak a csomópontot másoljuk át, majd a szöveges tartalmát a régi szövegtartalomtól függően határozzuk meg.

A következő animációban egy egyszerűbb példán keresztül mutatjuk be az egyes parancsok végrehajtási sorrendjének alapelveit: **Xsl működési mechanizmus demo animáció**. A XSLT nyelv parancsainak részletes leírását a következő fejezet adja meg.

3.7. Változók használata

Az XSLT transzformációk során előfordulhat, hogy olyan transzformációs formuláink vannak, melyben egy olyan közös tag is szerepel, amely önmagában is összetett, ezért sokszori ismétlése áttekinthetetlené teszi az XSLT programot. A hagyományos programozási nyelvekben ekkor egy rövidítést, szimbolikus nevet rendelnek ehhez a taghoz. Egy hasonló mechanizmus él az XSLT nyelvben is, ezek lesznek a változók (variable). A 'változó' elnevezés itt azonban kicsit zavaró, mivel ezen szimbólumok értéke nem módosulhat, értéke tehát nem változhat. Használatának indokai:

- beszédes jelölés az értékekre, pl. Pi a 3.1416 helyett,
- formulák áttekinthetővé tétele a részkifejezések kiemelésével,
- leírás egyszerűsítése az ismétlődő rész helyettesítésével.

Az XSLT nyelvben kétféle hatáskörű változó létezik. A globális változó a teljes XSLT programban felhasználható, a lokális változó csak a definíciós blokkjában. A változó létrehozásának utasítása:

```
<xsl:variable name="nev" as="adat_tipus"> ... </xsl:variable>
```

A változó értékét az elem szövegrészében vagy az elem 'select' elemjellemzőjénél lehet megadni. A változó akkor lesz globális hatáskörű, ha a definiálása a gyökér alatt történik. Különben lokális érvényű. A váltózókra a kifejezésekben a

```
$nev
```

formában lehet hivatkozni, ekkor értékükkel helyettesítődnek. A változók adattípusának megkötéséhez az XMLSchema-t lehet segítségül hívni, itt is az ott definiált adattípusokra lehet hivatkozni. A típuskijelölésnél az XMLSchema névtere egyértelműsit. A típuskijelölés menetét mutatja be a következő kis példa:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
   <xsl:variable name="dx" as="xs:integer">
```

A példában egy egész értékű, dx nevű változót hoztunk létre. A változókezelés tágabb kontextusát mutatja be következő példánk, melyben a bemenő adatállomány körök adatait adja meg a középpont és egy tetszőleges pont adatain keresztül:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx3.xsl" ?>

<korok>
   <kor>
      <kpont> <x>5</x><y>5</y></kpont>
      <pont> <x>10</x><y>12</y></pont>
   </kor>

   <kor>
      <kpont> <x>2</x><y>5</y></kpont>
      <pont> <x>8</x><y>12</y></pont>
   </kor>

</korok>
```

A transzformációs XSLT programnak ebből olyan alakot kell előállítania, melyben a kört leíró adatokat a területtel egészítjük ki. Ehhez előbb kiszámoljuk a sugárnégyzetet, majd a területet. A számításoknál a változókat a következő célokra használjuk fel:

- a pi érték tárolása,
- részeredmények tárolása.

A konverziós program felépítését az alábbi lista mutatja be:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="korok">
   <xsl:element name="korok">
   <xsl:apply-templates/>
   </xsl:element>
</xsl:template>

  <xsl:variable name="pi" as="xs:float"> 3.1416 </xsl:variable>
<xsl:template match="kor">
   <xsl:copy>
        <xsl:copy-of select="kpont"></xsl:copy-of>
        <xsl:copy-of select="pont"></xsl:copy-of>

       <xsl:element name="terulet">
```

```
          <xsl:variable name="dx" as="xs:integer">
              <xsl:value-of select="number(kpont/x)  - number(pont/x)"/>
          </xsl:variable>
          <xsl:variable name="dy" as="xs:integer">
            <xsl:value-of select="number(kpont/y) - number(pont/y)"/>
          </xsl:variable>
          <xsl:variable name="d" as="xs:float">
            <xsl:value-of select="($dx * $dx  + $dy * $dy)  * $pi"/>
          </xsl:variable>
          <xsl:value-of select=" $d"></xsl:value-of>
      </xsl:element>
   </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

A példában felhasználtuk egyrészt a 'number' függvényt, amely egy szöveges értéket numerikus alakra konvertál, másrészt azt a tényt, hogy numerikus műveleteket nem az XSLT maga, hanem a beágyazható XPath támogat. Emiatt, például egy

```
 <xsl:variable name="d" as="xs:float">
   number($dx * $dx  + $dy * $dy)
 </xsl:variable>
```

kifejezést érvénytelennek tekint a feldolgozó. A transzformációval kapott eredmény dokumentum:

```
 <?xml version="1.0" encoding="UTF-8"?>

 <korok>
  <kor>
    <kpont> <x>5</x><y>5</y></kpont>
    <pont> <x>10</x><y>12</y></pont>
    <terulet>232.4784</terulet>
  </kor>

  <kor>
    <kpont> <x>2</x><y>5</y></kpont>
    <pont> <x>8</x><y>12</y></pont>
    <terulet>267.03598</terulet>
  </kor>

</korok>
```

http://www.xml.com/pub/a/2001/05/07/xsltmath.html
http://www.oracle.com/technology/pub/articles/wang_xslt.html

3.8. Csoportképzés

Az adatbázis kezelésben megszokott dolog a rekordok csoportosítása és aggregált adatok képzése. Az XSLT is megvalósította ezt a funkciót a 2-es verziójában. A csoportképzésnél meg kell adni a feldolgozandó elemeket és a csoportképzési kifejezést. Emellett még számos további működési paraméter is beállítható. A feldolgozás

során minden csoportnál megáll, s lehetőséget ad csoport-szintű aggregációra és elem szintű részletezésre. A relációs modelltől eltérően itt tehát a csoport alkotó tagjai is elérhetők, külön-külön is. A csoportképzés parancsa:

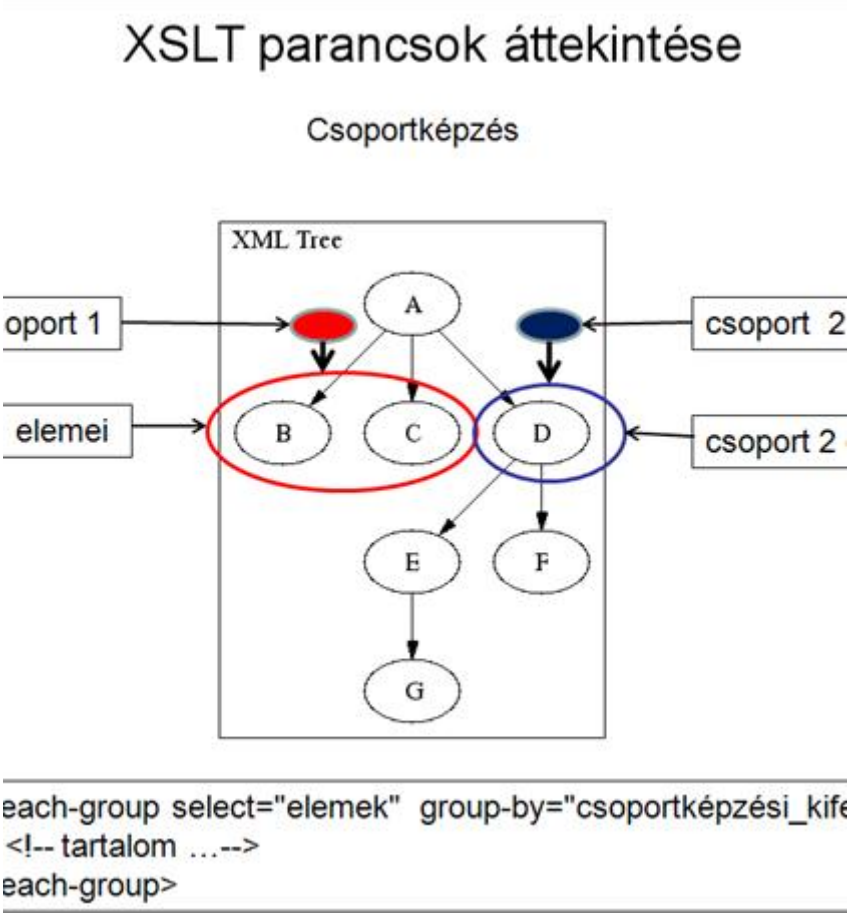
```
<xsl:for-each-group select="elemek"  group-by="csoportképzési_kifejezes" >...<..>
```

A feldolgozó magban két speciális szimbólum használható a csoport elemeinek elérésére:

- current-group: az aktuális csoport elemeinek szekvenciája,
- current-grouping-key : az aktuális csoporthoz tartozó csoportképzési kifejezés.

Hivatkozáskor a current_group szimbólum mögé a csoportbeli elemeket kell érteni, értéke ezért nemcsak skalár lehet, hanem lista is. Az aggregációk végrehajtására felhasználhatók az XSLT-XPath numerikus, aggregációs függvényei. A rendelkezésre álló függvények:

- avg()
- sum()
- max()
- min()
- count()



13. Csoportképzés XSLT parancsa

A függvények argumentuma csomópontok, kifejezések halmaza. A dokumentumban fellelhető 'vizsga' csomópontok darabszámának kiírását mutatja be az alábbi parancsállomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xsl:template match="/">
  <xsl:copy>
    <xsl:element name="db">
      <xsl:value-of select="count(//vizsga)"></xsl:value-of>
    </xsl:element>
  </xsl:copy>

 </xsl:template>
</xsl:stylesheet>
```

A következő példában a vizsgajegyek dokumentumához készítünk összesítő jelentést. Az adatforrás dokumentum:

```
?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx4.xsl" ?>

<vizsgak>
  <vizsga>
    <targy> Matek</targy> <diak> K011</diak><jegy> 2</jegy>
  </vizsga>

  <vizsga>
    <targy> Nemet</targy> <diak> K071</diak><jegy> 3</jegy>
  </vizsga>

  <vizsga>
    <targy> Matek</targy> <diak> K071</diak><jegy> 2</jegy>
  </vizsga>

  <vizsga>
    <targy> Nemet</targy> <diak> K101</diak><jegy> 1</jegy>
  </vizsga>

  <vizsga>
    <targy> Matek</targy> <diak> K271</diak><jegy> 3</jegy>
  </vizsga>

</vizsgak>
```

A konverziós állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:template match="/">
    <xsl:copy>
       <xsl:apply-templates/>
     </xsl:copy>

  </xsl:template>

  <xsl:template match="vizsgak">
    <xsl:element name="osszesito">
      <xsl:for-each-group select="vizsga" group-by="targy" >
        <xsl:element name="targy" >
           <xsl:element name="nev">
             <xsl:value-of select=" current-grouping-key()"></xsl:value-of>
           </xsl:element>
        <xsl:element name="atlag">
          <xsl:value-of select="avg(current-group()/jegy)"></xsl:value-of>
        </xsl:element>
        </xsl:element>
      </xsl:for-each-group>
     </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Az eredményül kapott összesítő:

```
<?xml version="1.0" encoding="UTF-8"?>
<osszesito>
  <targy><nev> Matek</nev><atlag>2.3333333333333335</atlag></targy>
  <targy><nev> Nemet</nev><atlag>2</atlag></targy>
</osszesito>
```

A csoportképzésnél az alap működési módban a feldolgozó dokumentum sorrendben járja be a csoportok alkotó elemeit. Akkor fog új csoport létrejönni, ha egy új érték jelenik meg. Emiatt a csoportok előfordulási sorrendje különbözhet a csoportképzési kifejezés szerinti sorrendtől. Ha a csoportokat a csoportképzési kifejezés szerint szeretnénk megjeleníteni, akkor egy rendezést előíró parancsot (sort) kell megadni a 'for-each-group' első gyerekelemeként. A rendezett kiíratásra ad példát az alábbi minta. Forrás állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx4.xsl" ?>
<vizsgak>
  <vizsga>
     <targy> Matek</targy> <diak> K011</diak><jegy> 2</jegy>
  </vizsga>
  <vizsga>
     <targy> Angol</targy> <diak> K271</diak><jegy> 3</jegy>
  </vizsga>
  <vizsga>
     <targy> Nemet</targy> <diak> K071</diak><jegy> 3</jegy>
  </vizsga>
```

```
  <vizsga>
     <targy> Matek</targy> <diak> K071</diak><jegy> 2</jegy>
  </vizsga>
  <vizsga>
     <targy> Nemet</targy> <diak> K101</diak><jegy> 1</jegy>
  </vizsga>
  <vizsga>
     <targy> Matek</targy> <diak> K271</diak><jegy> 3</jegy>
  </vizsga>
  <vizsga>
     <targy> Angol</targy> <diak> K171</diak><jegy> 2</jegy>
  </vizsga>
</vizsgak>
```

Az XSL parancsállomány:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:template match="/">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>

  </xsl:template>

  <xsl:template match="vizsgak">
    <xsl:element name="osszesito">
      <xsl:for-each-group select="vizsga" group-by="targy"   >
        <xsl:sort select="targy"/>
        <xsl:element name="targy" >
          <xsl:element name="nev">
            <xsl:value-of select=" current-grouping-key()"></xsl:value-of>
          </xsl:element>
        <xsl:element name="atlag">
          <xsl:value-of select="avg(current-group()/jegy)"></xsl:value-of>
        </xsl:element>
        </xsl:element>
      </xsl:for-each-group>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

A kapott eredménydokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<osszesito>
  <targy><nev> Angol</nev><atlag>2.5</atlag></targy>
  <targy><nev> Matek</nev><atlag>2.3333333333333335</atlag></targy>
  <targy><nev> Nemet</nev><atlag>2</atlag></targy>
</osszesito>
```

A szokásos csoportképzési technika mellett lehetőség van olyan módszer kérésére is, amikor a feldolgozó motor nem gyűjti össze a különböző helyen lévő, de azonos csoportképzési kifejezéshez tartozó elemeket. A dokumentum sorrendben haladva, minden olyan csoportképzési kifejezés érték, amely különbözik az előző elemhez tartozó értéktől, új csoport nyitását eredményezi, emiatt egy csoportképzési kifejezésérték többször is előfordulhat. Ezen feldolgozási mód esetében a 'group-by' jellemző helyett a 'group-adjacent' jellemzőt kell szerepeltetni. A mintaként vett feladat módosított alakja:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xsl:template match="/">
   <xsl:copy>
     <xsl:apply-templates/>
   </xsl:copy>
 </xsl:template>

 <xsl:template match="vizsgak">
    <xsl:element name="osszesito">
       <xsl:for-each-group select="vizsga"  group-adjacent="targy"   >
          <xsl:element name="targy" >
             <xsl:element name="nev">
                <xsl:value-of select=" current-grouping-key()"></xsl:value-of>
             </xsl:element>
             <xsl:element name="atlag">
               <xsl:value-of select="avg(current-group()/jegy)"></xsl:value-of>
             </xsl:element>
          </xsl:element>
       </xsl:for-each-group>
    </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

Az előálló eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>
<osszesito><targy><nev> Matek</nev><atlag>2</atlag></targy>
  <targy><nev> Angol</nev><atlag>3</atlag></targy>
  <targy><nev> Nemet</nev><atlag>3</atlag></targy>
  <targy><nev> Matek</nev><atlag>2</atlag></targy>
  <targy><nev> Nemet</nev><atlag>1</atlag></targy>
  <targy><nev> Matek</nev><atlag>3</atlag></targy>
  <targy><nev> Angol</nev><atlag>2</atlag></targy>
</osszesito>
```

A csoportképzés további lehetséges megvalósítási formája a 'group-starting-with' jellemző használatával hívható elő. Ekkor az elemek dokumentum sorrendben kerülnek feldolgozásra, és amikor a megadott elemtípus valamely előfordulásához érnek, egy új csoportot képeznek, függetlenül az elem értékétől. A 'group-ending-

with' opció esetén a kijelölt elemtípus a futó csoport végét jelzi. Így tehát négyféle csoportképzési módszer közül lehet választani, a csoportképzési parancsban az alábbi négy jellemző közül egynek kell előfordulnia:

- group-by,
- group-adjancent,
- group-starting-with,
- group-ending-with.

3.9. Paraméterezett formulák, függvények

A konverziós állományok rugalmasságának egyik fontos eszköze a paraméterezhető transzformációs sémák mechanizmusa. Ezt azt jelenti, hogy a transzformációs sémát (template) nem mintához kötjük, hanem közvetlenül meghívhatjuk, amihez egy azonosító névvel kell rendelkeznie. Ekkor tehát a séma nem 'match' hanem egy 'name', azonosító nevet kijelölő jellemzővel rendelkezik:

```
<xsl:template  name="azonosito_nev" ...> ... </xsl:template>
```

A névvel rendelkező sémát, az alprogramokhoz hasonlóan egy külön paranccsal, a nevük megadása mellett lehet aktivizálni. Az aktivizálás utasítása:

```
<xsl:call-template name="azonosito_nev"> .... </xsl:call-template>
```

Az előbb az alprogramokhoz hasonlítottuk az névvel ellátott sémákat. A hasonlóság abban is megnyilvánul, hogy itt is lehet paramétereket átadni híváskor. A formális paraméterek kijelölése a sémában a 'param' paranccsal történik:

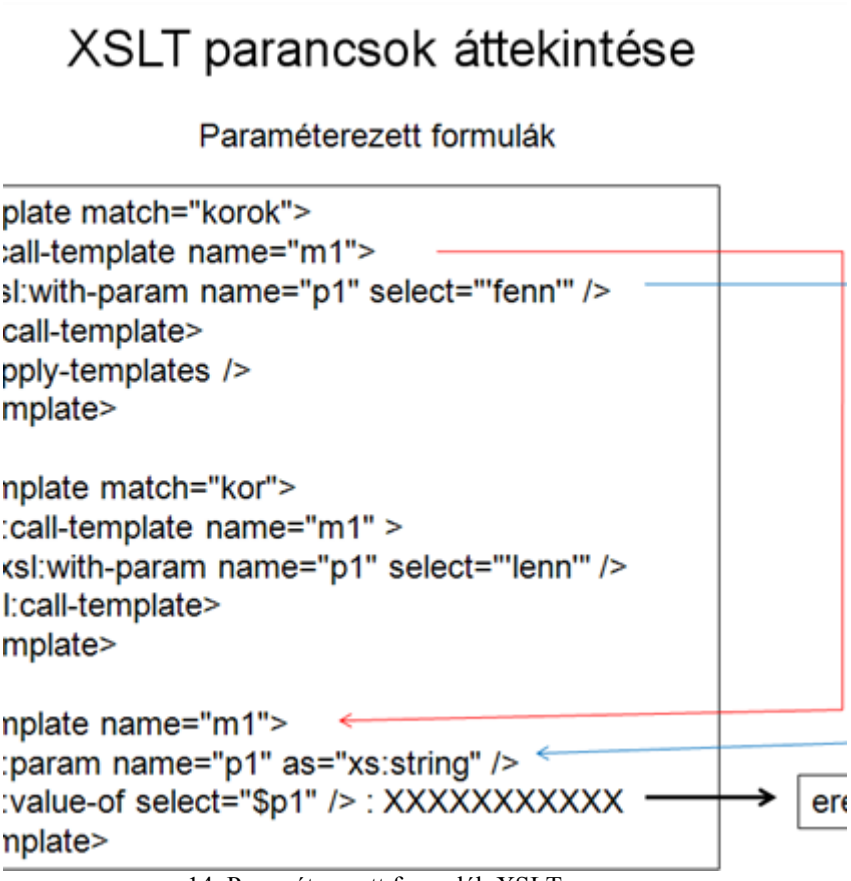
```
<xsl:param name="parameter_nev" ...> ....  </xsl:param>
```

A paraméter használatára, jellegére vonatkozólag több különböző beállítás tartozik. A legfontosabb jellemzők, amelyek a 'param' elemnél előfordulhatnak:

- as : a paraméter adattípusának megadása,
- select : az alapértelmezési paraméterérték, ha nem szerepelne aktuális paraméter,
- required : a paramétert kötelező megadni a híváskor.

A híváskor az aktuális paramétereket a 'call-template' elem gyerekeként szerepeltetik, a paraméterátadás utasítása a 'with-param' elem, melynek alakja:

```
<xsl:with-param name="parameter_nev"  select="ertek"> .. </ >
```



14. Paraméterezett formulák XSLT parancsa

A paraméterezett sémák használatának elemeit fogja össze az alábbi példa, melyben egy 'm1' nevű séma szerepel, melyet különböző aktuális paraméterértékkel aktivizálunk a 'korok' és 'kor' elemeknél. A konverziós állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="korok">
   <xsl:call-template name="m1">
      <xsl:with-param name="p1" select="'fenn'" />
   </xsl:call-template>
   <xsl:apply-templates />
 </xsl:template>

 <xsl:template match="kor">
    <xsl:call-template name="m1" >
       <xsl:with-param name="p1" select="'lenn'" />
    </xsl:call-template>
 </xsl:template>

 <xsl:template name="m1">
     <xsl:param name="p1" as="xs:string" />
     <xsl:value-of select="$p1" /> : XXXXXXXXXXX
```

```
 </xsl:template>
 </xsl:stylesheet>
```

A létrehozott 'm1' sémának egy, 'p1' névvel rendelkező paramétere van, amely szöveges adattípusú. A fenti állományt az alábbi bemenő dokumentumra futtatjuk le:

```
?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx6.xsl" ?>

<korok>
 <kor>
    <kpont> <x>5</x><y>5</y></kpont>
    <pont> <x>10</x><y>12</y></pont>
 </kor>

 <kor>
    <kpont> <x>2</x><y>5</y></kpont>
    <pont> <x>8</x><y>12</y></pont>
 </kor>

</korok>
```

A kapott eredménydokumentum, amely most egy érvénytelen XML dokumentum, tartalma :

```
<?xml version="1.0" encoding="UTF-8"?>

  fenn : XXXXXXXXXXX

  lenn : XXXXXXXXXXX

  lenn : XXXXXXXXXXX
```

A mechanizmus működésére még egy további példát veszünk, melyben a forrásdokumentum termékek leírását tartalmazza, ahol a termék lehet CD vagy könyv. A kimeneti dokumentumban egy szöveges leírás szerepel a termékeknél, ahol a szövegrész formátuma attól függ, hogy milyen terméktípusnál járunk. A forrásállomány tartalma:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx7.xsl" ?>

<termekek>
 <termek>
   <kod>IUC</kod> <kategoria>CD</kategoria> <ar>1233</ar>
 </termek>
 <termek>
    <kod>HH1</kod> <kategoria>B</kategoria> <ar>833</ar>
 </termek>
 <termek>
    <kod>OT7</kod> <kategoria>CD</kategoria> <ar>1433</ar>
```

```
  </termek>
  <termek>
     <kod>DGW</kod> <kategoria>B</kategoria> <ar>454</ar>
  </termek>
  <termek>
     <kod>JIU</kod> <kategoria>CD</kategoria> <ar>760</ar>
  </termek>

</termekek>
```

A konverziós állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"     >

<xsl:template match="/">
   <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
 </xsl:template>

 <xsl:template match="*">
     <xsl:copy>
       <xsl:apply-templates/>
     </xsl:copy>
 </xsl:template>

  <xsl:template match="termek">
     <xsl:copy>
       <xsl:call-template name="m2">
         <xsl:with-param name="p2" select="kategoria"></xsl:with-param>
        </xsl:call-template>
     </xsl:copy>
  </xsl:template>

  <xsl:template name="m2">
     <xsl:param name="p2" as="xs:string"></xsl:param>
     Van egy
     <xsl:choose>
        <xsl:when test="$p2 ='CD'"> lemezunk</xsl:when>
        <xsl:otherwise> konyvunk</xsl:otherwise>
     </xsl:choose>
     , melynek ara <xsl:value-of select="ar"/>
  </xsl:template>

</xsl:stylesheet>
```

A kapott eredmény dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<termekek>
```

```
  <termek>
     Van egy
      lemezunk
     , melynekl ara 1233
  </termek>
  <termek>
     Van egy
      konyvunk
     , melynekl ara 833
  </termek>
  <termek>
     Van egy
      lemezunk
     , melynekl ara 1433
  </termek>
  <termek>
     Van egy
      konyvunk
     , melynekl ara 454
  </termek>
  <termek>
     Van egy
      lemezunk
     , melynekl ara 760
    </termek>
</termekek>
```

A paraméterezett sémák az egyes csomópontok komplett feldolgozására adnak rugalmas keretet. Sokszor viszont csak kisebb kódrészlet ismétlésére van szükség, mint például egy kódszám ellenőrzése vagy egy számjegy szöveges alakra történő konvertálása. Ezen feladatok ellátása a függvények szolgálnak. A függvény definiálása hasonlít a névvel ellátott sémákra, viszont meghívása minden olyan kifejezésből megtörténhet, ahol ezt az adattípus illesztés megengedi. A függvény nevére érvényes egy olyan megkötés, hogy a névnek névtérrel kiterjesztett alakúnak kell lennie. A függvény visszatérési értéke a függvényben a végrehajtása során előállított, kiírt érték, nincs külön visszatérési érték kijelölő utasítás. A függvény létrehozásának alakja:

```
<xsl:function name="nevter:nev" ... > ...   </xsl:function>
```

A következő példa egy üdvözlő szöveget előállító függvényt definiál. Az 'udv' nevű függvény a 'kl' névhez rendelt névtérhez tartozik:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:kl="kl1.kl1">
  ..
  <xsl:function name="kl:udv" as="xs:string">
    Hello vilag!
  </xsl:function>
  ...
</xsl:stylesheet>
```

A függvény meghívása a nevével történik, a hívás XPath kifejezésben is szerepelhet. A következő példában egy numerikus értékkel visszatérő függvény definiálását és hívását láthatjuk:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:kl="kl1.kl1">
  ...
  <xsl:value-of select="1+kl:f1()"/>
  ...
   <xsl:function name="kl:udv" as="xs:integer">
     <xsl:number  value="2" />
   </xsl:function>
  ...

</xsl:stylesheet>
```

A példában szereplő 'number' elem konverziós utasításként értelmezendő, amely numerikus típusra alakítja át a megadott kifejezést. A kifejezés lehet csomópont szövegtartalma is. A függvény hívása a 'value-of' paranccsal történik, a példánkban mindig a 3-as értéket eredményezve. A függvényekhez is köthető paraméterátadás. A formális paraméterek megadása a már említett 'param' elemmel történik, az aktuális paramétereknek a hívási függvénynevet követő zárójelben kell szerepelniük. Az előző, a forrásdokumentumot feldolgozó parancsállományt úgy alakítjuk át, hogy a kimeneti dokumentumban az ár mellett megadjuk, hogy ez olcsó vagy drága termék. Az értékelést egy függvény végzi, melynek két bemenő paramétere van, a kategória és az ár. A konverziós állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:kl="kl1.kl1">

 <xsl:template match="/">
   <xsl:copy>
     <xsl:apply-templates/>
   </xsl:copy>
 </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="termek">
    <xsl:copy>
      <xsl:call-template name="m2">
        <xsl:with-param name="p2" select="kategoria"></xsl:with-param>
      </xsl:call-template>
    </xsl:copy>
  </xsl:template>

  <xsl:template name="m2">
     <xsl:param name="p2" as="xs:string"></xsl:param>
```

```
    Van egy
    <xsl:choose>
      <xsl:when test="$p2 ='CD'"> lemezunk</xsl:when>
      <xsl:otherwise> konyvunk</xsl:otherwise>
    </xsl:choose>
    , melynekl ara <xsl:value-of select="ar"/> , <xsl:value-of select="kl:f1(kategoria, ar)"/>
  </xsl:template>

  <xsl:function name="kl:f1" as="xs:string">
    <xsl:param name="pk"></xsl:param>
    <xsl:param name="pa"></xsl:param>
    <xsl:if test="$pk='CD'">
      <xsl:choose>
        <xsl:when test="$pa &#60; 1100"> olcso termek </xsl:when>
        <xsl:otherwise> draga termek</xsl:otherwise>
      </xsl:choose>
    </xsl:if>
    <xsl:if test="$pk = 'B'">
      <xsl:choose>
        <xsl:when test="$pa &#60; 600"> olcso termek </xsl:when>
        <xsl:otherwise> draga termek</xsl:otherwise>
      </xsl:choose>
    </xsl:if>

  </xsl:function>

</xsl:stylesheet>
```

A függvények segítségével lehet kikerülni, legalábbis részben az XSLT nyelv azon korlátját, hogy a változók konstans volta miatt nem lehet taxatív ciklusokat szervezni. Vegyünk például egy sima ciklust, amely egyesével léptet 1 és 8 között. Ezt a feladatot nem oldhatjuk meg XSLT-ben egy ciklusváltozó léptetésével, hiszen a változó értéke nem módosulhat. A megoldás a függvények megfelelő, rekurzív használatában rejlik. A megoldás magva az a lehetőség, hogy a ciklusmagot egy függvény mögé tesszük le, és ezt a függvényt hívjuk meg többször, rekurzívan. Minden hívásnál az aktuális paramétert léptetjük a szükséges mértékkel, így minden hívás egy-egy ciklusmag végrehajtásnak felel meg a különböző ciklusváltozó értékek mellett. A rutin belsejében ellenőrizzük, hogy szükség van-e további rekurzív feldolgozásra. Az alábbi kódrészlet egy megoldást mutat be a kijelölt feladatra:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:kl="kl1.kl1">

 <xsl:template match="/">
    <xsl:copy>
      <xsl:element name="fo">
        <xsl:value-of select="kl:f1(1,8)"></xsl:value-of>
      </xsl:element>
    </xsl:copy>
 </xsl:template>

 <xsl:function name="kl:f1">
  <xsl:param name="p"  as="xs:integer"/>
  <xsl:param name="pm"  as="xs:integer"/>
  <xsl:if test="$p &#60; $pm">
    <xsl:value-of select="$p"/>X
```

```
    <xsl:value-of select="kl:f1($p+1, $pm)"/>
   </xsl:if>
  </xsl:function>

</xsl:stylesheet>
```

Az eredményül kapott dokumentum:

```
 <?xml version="1.0" encoding="UTF-8"?>
<fo>
    1X
    2X
    3X
    4X
    5X
    6X
    7X
</fo>
```

Az iteráció segítségével összetettebb számítások is elvégezhetők. Példaként vegyünk egy olyan függvényt, amely hiányzik az alap XPath-XSLT készletből, az 'exp' exponenciális értéket megadó matematikai függvényt. A függvény implementációja az ismert iterációs, közelítő formulán alapszik:

$$exp(z) = e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + ...$$

A kódoláshoz a ciklus alapú felírást át kell alakítani rekurzív alakra, ahol az aktuális tagot ki lehet fejezni az előző szint megfelelő értékéből. A rekurzív formula XSLT kódolása:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:kl="kl1.kl1">

 <xsl:template match="/">
   <xsl:copy>
     <xsl:element name="fo">
       <xsl:value-of select="kl:exp(1)"></xsl:value-of>
     </xsl:element>
   </xsl:copy>
 </xsl:template>

<xsl:function name="kl:exp" as="xs:float">
   <xsl:param name="x" as="xs:float"/>
   <xsl:value-of select="1.+kl:exp2($x,1,1,1,8)"  />
</xsl:function>

 <xsl:function name="kl:exp2"  as="xs:float">
  <xsl:param name="x"  as="xs:float"/>
  <xsl:param name="szamlalo" as="xs:float"/>
  <xsl:param name="nevezo" as="xs:float"/>
  <xsl:param name="i" as="xs:integer"/>
  <xsl:param name="imax" as="xs:integer"/>
```

```
   <xsl:variable name="ujtag"  as="xs:float" select="$szamlalo*$x  div ($nevezo*$i)"  />
   <xsl:choose>
     <xsl:when test="$i &#60; $imax">
          <xsl:value-of  select="$ujtag+ kl:exp2($x,$szamlalo*$x, $nevezo*$i,$i+1,$imax)" />
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select="$ujtag"  />
     </xsl:otherwise>
   </xsl:choose>

 </xsl:function>

</xsl:stylesheet>
```

A kódban a 'div' az osztás operátora. A felhasználó a külső 'exp' függvényt hívja meg, amely megfelelő paraméterezéssel az 'exp2'-nek adja át a vezérlést. A belső függvény paraméterei:

- x értéke
- számláló értéke az akutuális tagnál
- nevező értéke az aktuális tagnál
- szint, lépésjelző
- maximális lépésszám

A példánkban 8 tagig számoljuk ki exp(1) értékét. A megjelenő eredmény dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo>
 2.718279
</fo>
```

3.10. Egyéb elemek

Abban az esetben, ha a létrehozandó dokumentumban több elemnél is ugyanazon elemjellemzők fordulnak elő ismétlődően, akkor az 'attribute-set' utasítás lehetőséget ad a parancsok egyszerűsítésére. Ezen parancs a jellemzők egy halmazát hozza létre, mely több jellemzőt fog össze az értékeikkel együtt, s egy azonosító név kötődik a halmazhoz. Az elemek létrehozásakor a névvel betölthető a halmaz, s ekkor az elem felveszi a halmazhoz tartozó összes jellemzőt. A halmaz beépítését az elembe a 'use-attribute-sets' utasítás hajtja végre. Ha a halmazban megadott jellemző értéke nem megfelelő egy adott elemnél, akkor annak értékét a már ismert 'attribute' utasításon keresztül lehet módosítani. A következő példa egy 's1' nevű halmazt definiál, melyet a 'fopont' elemnél használunk fel:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0" >

 <xsl:template match="/">
   <xsl:element name="fopont" xsl:use-attribute-set="s1" >
      <xsl:attribute name="kor" select="11" />
Hello
    </xsl:element>
```

```
</xsl:template>

 <xsl:attribue-set name="s1">
   <xsl:attribute name="nev" select="'bodri'" />
   <xsl:attribute name="kor" select="8" />
 </xsl:attribute-set>

</xsl:stylesheet>
```

A példában a 'kor' jellemző alapértelmezési értéke lett átírva 11-re. A kapott eredménydokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
 <fopont nev="bodri"   kor="11" >
   Hello
 </fopont>
```

A szekvencia több elemi érték listáját jelenti. A 'sequence' operátor elemeket fűz össze egy listába. Ez a funkció felhasználható többek között a változók értékadásánál is. A változóknál a felvett érték ugyanis nemcsak skalár szöveg vagy numerikus érték lehet, hanem tetszőleges szekvencia, lista is. A listát kerek zárójelben, a tagok felsorolásával lehet megadni. A következő példában egy 'p1' azonosítójú változót hozzunk létre, amely egy négyelemű értéklistát tartalmaz. A kódrészlet további soraiban előbb kiíratjuk az alkotó elemek összegét a 'sum()' aggregációs függvénnyel, majd egyenként is feldolgozzuk a lista elemeit a 'for-each' utasítás segítségével:

```
<xsl:template match="....">

  <xsl:variable name="p1" as="xs:integer*">
    <xsl:sequence select="(1,2,3,8)"/>
    <xsl:sequence select="(10,20)"/>
  </xsl:variable>
  <xsl:value-of select="sum($p1)"/>
  <xsl:for-each select="$p1">  <xsl:value-of select="."/></xsl:for-each>

 </xsl:template>
```

Mint a példában is látható, több egymás utáni 'sequence' utasítás egy eredő szekvenciát hoz létre.

A hatékony és rugalmas szövegelemzéshez megvalósították a reguláris kifejezésre épülő szövegvizsgálatot is. A már megismert REGEX szabályoknak megfelelő illesztés vizsgálatot az 'analyze-string' utasítással lehet megvalósítani, a parancs formátuma:

```
<xsl:analyze-string  select="szöveg" regex="minta">
   <!-- feldolgozás -->
</xsl:analyze-string>
```

A feldolgozási részben lehet megadni, hogy az illeszkedő, vagy nem illeszkedő szövegrésszel mit kell tenni. A feldolgozáshoz két parancs áll rendelkezésre. Az illeszkedő szövegrészt a

```
<xsl:matching-substring>
  <!-- műveletek -->
</xsl:matching-substring>
```

parancs írja le. Minden illeszkedőnek talált szövegrész egy-egy elemként kezelődik a feldolgozásnál. A nem illeszkedő szövegrészeket a

```
<xsl:non-matching-substring>
  <!-- műveletek -->
</xsl:non-matching-substring>
```

elemmel lehet feldolgozni. Példaként keressük a 'leiras' nevű elemből a számokat:

```
<xsl:template match="//leiras">
  <xsl:analyze-string  select="." regex="[0-9]+">
     <xsl:matching-substring>
        <xsl:value-of select="." />  <br/>
     </xsl:matching-substring>
  </xsl:analyze-string>
</xsl:template>
```

A kulcs mechanizmussal lehetőség nyílik az egyes csomópontok gyorsabb, közvetlen elérésére. Működése a következő lépésekből áll. Elsőként egy kulcs indexet kell létrehozni a 'key' utasítás segítségével az elemekhez. Ha az megtörtént, akkor már felhasználható ez az index az elemek keresésére, melyhez a 'key()' hivatkozási függvényt kell meghívni. A 'key' utasítás fontosabb paraméterei:

- a kulcsindex neve,
- indexelt csomópontok listája,
- a kulcsérték meghatározása a csomópontoknál.

Az elemkeresésnél a csomópontot meghatározó függvény paraméterei:

- kulcsindex neve,
- kulcsérték.

Az eredmény az illeszkedő elemek halmaza, melyből a szokásos útvonal kifejezésekkel tetszőleges részeket emelhetünk ki. A következő példában az alábbi alapdokumentumból indulunk ki:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx10.xsl" ?>

<termekek>
```

```
  <termek>
    <kod>IUC</kod> <kategoria>CD</kategoria> <ar>1233</ar> <db>23</db>
  </termek>
  <termek>
     <kod>HH1</kod> <kategoria>B</kategoria> <ar>833</ar><db>12</db>
  </termek>
  <termek>
     <kod>OT7</kod> <kategoria>CD</kategoria> <ar>1433</ar><db>12</db>
  </termek>
  <termek>
     <kod>DGW</kod> <kategoria>B</kategoria> <ar>454</ar><db>56</db>
  </termek>
  <termek>
     <kod>JIU</kod> <kategoria>CD</kategoria> <ar>760</ar><db>31</db>
  </termek>

</termekek>
```

A forrásban a 'termek' csomópont lesz a keresendő, indexelt egyed és indexkulcsként a 'kod' mező szolgál. Az indexfelépítést és keresést megvalósító XSLT program:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

  <xsl:key name="K1" match="termek" use="kod" />

  <xsl:template match="/">
    <xsl:copy>
      <xsl:element name="fo">
        <xsl:value-of select="key('K1','HH1')/ar"></xsl:value-of>
       </xsl:element>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Az elkészült eredménydokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
 <fo>
   833
 </fo>
```

Listabeli elemek sorszámértékeinek kijelzési formátumát állítja be a 'number' utasítás. Eredménye egy olyan szövegcsomópont, amely a formátumozott alakot tartalmazza. A konverziós utasítás fontosabb paraméterei:

- count: a leszámlálandó csomópontok kijelölése,
- level: a számlálás szintjelzője, lehet egyszintű (single) és többszintű (multiple),
- from: a számlálás kezdetét kijelölő kifejezés,

- format: a sorszám megjelentési formátuma, lehetséges értékei:
  - 1: normál számjegyek,
  - a: betuk (a,b,c,...),
  - A: nagybetűk,
- value: közvetlen értékmegadás,
- lang: nyelv kijelölése,
- grouping-separator: a számjegyek csoportosítási karaktere,
- grouping-size: a csoportosítás nagysága, mennyi karakter kerüljön egy csoportba.

Példaként vegyük az alábbi induló dokumentumot:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx10.xsl" ?>

<termekek>
 <termek>
   <kod>IUC</kod> <kategoria>CD</kategoria> <ar>1233</ar> <db>23</db>
   <vevo>Peter</vevo> <vevo>Zoli</vevo>
 </termek>
 <termek>
   <kod>HH1</kod> <kategoria>B</kategoria> <ar>833</ar><db>12</db>
   <vevo>Anna</vevo>
 </termek>
 <termek>
   <kod>OT7</kod> <kategoria>CD</kategoria> <ar>1433</ar><db>12</db>
   <vevo>Peter</vevo><vevo>Gabi</vevo>
 </termek>
 <termek>
   <kod>DGW</kod> <kategoria>B</kategoria> <ar>454</ar><db>56</db>
   <vevo>Tomi</vevo><vevo>Anna</vevo><vevo>Peter</vevo>
 </termek>
 <termek>
   <kod>JIU</kod> <kategoria>CD</kategoria> <ar>760</ar><db>31</db>
   <vevo>Tomi</vevo>
 </termek>

</termekek>
```

A fenti listát úgy alakítjuk át, hogy csak a termékkódot és a vevőnevet íratjuk ki. A kiíratásnál egy többszintű sorszámot rendelünk a kiírt elemekhez. A konverziós állomány:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

 <xsl:key name="K1" match="termek" use="kod" />

 <xsl:template match="/">
  <xsl:copy>
```

```
      <xsl:element name="fo">
        <xsl:for-each select="termekek/termek">
          <xsl:number count="vevo | termek" format="1.1"  level="multiple"/>
          <xsl:value-of select="kod"></xsl:value-of><br/>
          <xsl:for-each select="vevo">
             <xsl:number count="vevo | termek "  from="termekek"
             format="1.1"  level="multiple"></xsl:number>
              <xsl:value-of select="."/><br/>
           </xsl:for-each>
        </xsl:for-each>
      </xsl:element>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Mindkét sorszám kiíratásnál egyazon csomópontlistára hivatkoztunk. A feldolgozó figyeli a bevont elemek szülő-gyerek viszonyát, s ez alapján határozza meg a hierarchia szintet jelölő értékeket. A konverzió eredménye:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo>
   1 IUC<br/>
      1.1 Peter<br/>
      1.2 Zoli<br/>
   2 HH1<br/>
      2.1 Anna<br/>
   3 OT7<br/>
      3.1 Peter<br/>
      3.2 Gabi<br/>
   4 DGW<br/>
      4.1 Tomi<br/>
      4.2 Anna<br/>
      4.3 Peter<br/>
   5  JIU<br/>
     5.1Tomi<br/>
</fo>
```

A piaci szoftver termékeknél egyik fontos kritérium a megfelelő megjelenési formátum biztosítása. Egy vállalatnál igen fontos lehet például, hogy egy dátumérték minden dokumentumban azonos módon jelenjen meg. A különböző formátumok ugyanis téves értékekhez, bizonytalansághoz, többlet költségekhez vezethetnek. A programozó szempontjából az jelenti az egyik problémát, hogy egy értékhez, például dátumértékhez nagyon sok különböző megjelenési formátum tartozhat. Az XSL szerencsére rendelkezik ilyen formátum beállító lehetőséggel. A különböző alapadat típusokhoz más és más a konverziós függvény azonosító neve. Itt most a két legfontosabbat vesszük át, a dátum és szám konverziós függvényt. A dátumkonverziós függvények alakja:

```
format-date(dátum, formátum-maszk, nyelv, naptárkód)
```

A formátum-maszk szimbólumokkal írja le az eredmény sztring formátumát. Opcionálisan megadható az alkalmazandó nyelv és naptár kódja is. A formátum maszkban megadható fontosabb szimbólumok:

- Y: év,
- M: hónap számmal,

- MN: hónap betűvel,
- D: nap,
- d: a nap éven belüli sorszáma,
- F: a hét napja,
- W: a hét éven belüli sorszáma,
- w: a hét hónapon belüli sorszáma,
- H: óra,
- P: de és du jelzése,
- m: perc,
- s: másodperc.

A formátum szimbólumokat a '[]' jelek között kell megadni, ami a zárójeleken kívül van, az a megjelenítésbe közvetlenül átmegy. Néhány minta megjelenítési formátum:

- [Y]:[M]:[D] = 2007:08:23
- [Y]:[MN]:[D] = 2007:AUGUST:23
- [Y001]:[d0001] = 007:0204

A dátumok kezeléséhez kapcsolható az aktuális dátum vagy időpont lekérdezése. A kapcsolódó függvényhívások:

- current-date() : aktuális dátum,
- current-time() : aktuális idő.

A numerikus adatoknál a

```
format-number(szám, formátum-maszk)
```

A formátum-maszkban a legfontosabb szabvány szimbólumok:

- \# : számjegy,
- .: tizedespont,
- NaN : nincs adat,
- ,: csoport szeparátor.

Mivel a különböző nyelvekben más és más karakter lehet a szeparáló, jelölő elem, ezért szükség lehet a formátumok egyszerűsített kijelölésére is. A formátum definíciós parancs azonosító névvel ellátott formátum létrehozására alkalmas:

```
<xsl:decimal-format  name="név" decimal-separator="jel"  NaN="jel" ...>
```

Az így létrehozott formátum felhasználható a 'format-number' konverziós függvényben, ahol harmadik paraméterként szerepelhet:

```
format-number(szám, formátum-maszk,formátum-kód)
```

A következő példa egy saját formátum létrehozását és felhasználását mutatja be:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:decimal-format name="f1" decimal-separator="!"   NaN="nincs adat"/>
 <xsl:variable name="x">NaN</xsl:variable>

 <xsl:template match="/">
   <xsl:copy>
     <xsl:element name="fo">
       <xsl:value-of select="format-number($x,'A####!#A','f1')"></xsl:value-of><br/>
       <xsl:value-of select=" format-date( current-date(),'[Y001]:[MN]:[D] [d0001]' )"></xsl:value-of>
       <xsl:message >hello</xsl:message>
     </xsl:element>
   </xsl:copy>
 </xsl:template>

</xsl:stylesheet>
```

3.11. Külső XSLT transzformációs állomány bevonása

Az XSLT transzformációs programot is lehet modulárisan fejleszteni, azaz a különböző kódrészletek különböző forrásállományból jönnek össze. Egy adott transzformációs állományba egy másik transzformációs állomány tartalmát a

```
<xsl:include href="allomany">
```

utasítással lehet bevonni. A parancs hatására a külső állomány tartalmát áthozza az aktuális dokumentumba. Előfordulhat, hogy mind a külső és az aktuális állomány is tartalmaz egy-egy utasítást ugyanazon mintára. Ekkor a feldolgozási algoritmustól függ, hogy melyik érvényesül. Tesztjeinkben a később szereplő utasítás felülírja a korábban szereplőt. Például az alábbi esetben a gazda állomány utasítása érvényesül:

xx10.xsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:include href="xx11.xsl"/>

 <xsl:template match="/">
   <xsl:copy>
```

```
        <xsl:element name="fo">
          <xsl:apply-templates/>
        </xsl:element>
      </xsl:copy>
  </xsl:template>

  <xsl:template match="termek">
     HELLO termek!
  </xsl:template>
 </xsl:stylesheet>
```

xx11.xsl:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

<xsl:template match="termek">
    MI VAN termek!
 </xsl:template>

 </xsl:stylesheet>
```

Ha a beillesztés az aktuális dokumentum végére kerül, akkor a külső állományban megadott utasítás fog érvényesülni:

```
 ?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

 <xsl:decimal-format name="f1" decimal-separator="!"   NaN="nincs adat"/>
 <xsl:variable name="x">NaN</xsl:variable>

  <xsl:template match="/">
    <xsl:copy>
      <xsl:element name="fo">
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:copy>
  </xsl:template>

   <xsl:template match="termek">
     HELLO termek!
   </xsl:template>

   <xsl:include href="xx11.xsl"/>

 </xsl:stylesheet>
```

4. Feladatok, mintaprogramok

A soron következő feladatokban, az alábbi XML forrás állomány szerepel alapértelmezési forrásként

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="xx2.xsl" ?>

<autok>
 <auto rsz="r1">
    <tipus> Fiat </tipus> <ar> 21233 </ar><szin>piros</szin>
    <tulaj><nev>Zoli</nev><varos>Eger</varos></tulaj>
 </auto>
 <auto rsz="r4">
    <tipus> Skoda </tipus> <ar> 44233 </ar><szin>fehér</szin>
    <tulaj><nev>Peti</nev><varos>Miskolc</varos></tulaj>
 </auto>
 <auto rsz="r2">
    <tipus> Opel </tipus> <ar> 31233 </ar><szin>szürke</szin>
    <tulaj><nev>Tomi</nev><varos>Miskolc</varos></tulaj>
 </auto>
 <auto rsz="r7">
    <tipus> Opel </tipus> <ar> 21233 </ar><szin>kék</szin>
    <tulaj><nev>Sanyi</nev><varos>Miskolc</varos></tulaj>
 </auto>
</autok>
```

1. Feladat: Mi lesz az alábbi XSLT program eredménye?

```
<xsl:template match="*"  >
   UDV!
</xsl:template>
```

   Megoldás: Egy ÜDV! szöveg fog megjelenni.

2. Feladat: Állítsa elő az autók rendszámait visszaadó listát!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="auto"  >
     <xsl:value-of select="@rsz"/>
  </xsl:template>
</xsl:stylesheet>
```

3. Feladat: Állítsa elő az autók rendszámát és árát az ár szerinti sorrendben előállító listát!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
  <xsl:template match="/"  >
    <xsl:for-each select="//auto">
      <xsl:sort select="ar"/>
      <xsl:value-of select="@rsz"/>:<xsl:value-of select="ar"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

4. Feladat: Kérdezze le mennyi autó drágább mint 30000!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:value-of select="count ( //auto[number(ar) &#62; 30000])"></xsl:value-of>
  </xsl:template>

</xsl:stylesheet>
```

5. Feladat: Kérdezze le mennyi elemből áll a dokumentum!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
     elemdb=<xsl:value-of  select="count(//*)"/>
  </xsl:template>

</xsl:stylesheet>
```

6. Feladat: Állítsa elő az miskolci tulajdonosok autóinak rendszámait tartalmazó listát!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
 version="1.0">
 <xsl:template match="/"  >
    <xsl:for-each select="//auto[tulaj/varos/text()='Miskolc']">
       <xsl:value-of select="@rsz"/>
    </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

7. Feladat: Adja meg az autótípusokat és példányaik darabszámát példányszám szerint csökkenő sorrendben!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:element name="eredmeny">
     <xsl:for-each-group select="//auto" group-by="tipus/text()">
       <xsl:sort select="count( current-group())"/>
       <xsl:element name="tipus">
         <xsl:element name="tip"><xsl:value-of select="current-grouping-key()"/></xsl:element>
         <xsl:element name="db"><xsl:value-of select="count( current-group())"/></xsl:element>
       </xsl:element>
     </xsl:for-each-group>
   </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

8. Feladat: Adja meg városonként mennyi az ottani autók darabszáma és összára!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0" xmlns:kl="kl.xml">

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:element name="eredmeny">
      <xsl:for-each-group select="//auto" group-by="tulaj/varos/text()">
        <xsl:element name="varos">
          <xsl:element name="nev">
              <xsl:value-of select="current-grouping-key()"/></xsl:element>
          <xsl:element name="db">
              <xsl:value-of select="count( current-group())"/></xsl:element>
          <xsl:element name="összár">
```

```
                    <xsl:value-of select="sum( current-group()/ar)"/></xsl:element>
            </xsl:element>
          </xsl:for-each-group>
      </xsl:element>
   </xsl:template>


</xsl:stylesheet>
</xsl:stylesheet>
```

9. Feladat: Állítsa elő az autók rendszámát és árát az ár szerinti sorrendben tartalmazó XML állományt!

   Megoldás:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

    <xsl:template match="/"  >
      <xsl:element name="eredmeny">
      <xsl:for-each select="//auto">
         <xsl:sort select="ar"/>
          <xsl:element name="auto">
             <xsl:element name="rsz"><xsl:value-of select="@rsz"/></xsl:element> <xsl:copy-of
select="ar"/>
          </xsl:element>
      </xsl:for-each>
      </xsl:element>
   </xsl:template>

 </xsl:stylesheet>
```

10. Feladat: Vigye át az autókat egy olyan XML dokumentumba, ahol az elem típusa dragaauto vagy olcsoauto, attól függően, hogy az ár nagyobb vagy kisebb, mint 30000. Az elem szerkezet egyezzen meg az auto szerkezettel.

    Megoldás:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:element name="eredmeny">
      <xsl:for-each select="//auto">
        <xsl:choose>
          <xsl:when test="number(ar) &#62; 30000">
            <xsl:element name="dragaauto">
              <xsl:for-each select="* union @*">
                <xsl:copy-of select="."/>
              </xsl:for-each>
```

```
            </xsl:element>
          </xsl:when>
          <xsl:otherwise>
            <xsl:element name="olcsoauto">
              <xsl:for-each select="* union @*">
                <xsl:copy-of select="."/>
              </xsl:for-each>
            </xsl:element>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

11. Feladat: Adja meg az autótípusokat és példányaik darabszámát példányszám szerint csökkenő sorrendben!

Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:element name="eredmeny">
      <xsl:for-each-group select="//auto" group-by="tipus/text()">
        <xsl:sort select="count( current-group())"/>
        <xsl:element name="tipus">
          <xsl:element name="tip"><xsl:value-of select="current-grouping-key()"/></xsl:element>
          <xsl:element name="db"><xsl:value-of select="count( current-group())"/></xsl:element>
        </xsl:element>
      </xsl:for-each-group>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

12. Feladat: Készítsen egy autókat tartalmazó XML állományt, ahol az autó a CAR elembe kerül és az ID attribútumba kerül a rendszám, a TYPE attribútumba pedig a típus értéke!

Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:element name="eredmeny">
      <xsl:for-each select="//auto">
```

```
        <xsl:element name="CAR">
          <xsl:attribute name="ID" select="@rsz"/>
          <xsl:attribute name="TYPE" select="tipus"/>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

 </xsl:stylesheet>
```

13. Feladat: Készítsen egy nevesített template-et, mely egy megadott értéknél drágább autók rendszámait adja vissza egy XML részfában. Hívja meg az eljárást a 20000 értékkel!

Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">

  <xsl:template match="/"  >
    <xsl:call-template name="lista">
      <xsl:with-param name="minar">20000</xsl:with-param>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="lista">
    <xsl:param name="minar"/>
    <xsl:element name="eredmeny">
     <xsl:value-of select="$minar"/>
    <xsl:for-each select="root()//auto[number(ar) &#62; number($minar)]">
      <xsl:element name="auto">
        <xsl:value-of select="@rsz"></xsl:value-of>
      </xsl:element>
    </xsl:for-each>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

14. Feladat: Készítsen függvényt, amely kiszámolja a faktoriális értéket.

Megoldás:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0" xmlns:kl="kl.xml">

  <xsl:template match="/" >
    <xsl:value-of select="kl:fakt(6)"/>
  </xsl:template>
```

```
 <xsl:function name="kl:fakt">
   <xsl:param name="N"/>

   <xsl:choose>
     <xsl:when test="$N &#62; 1">
       <xsl:variable name="M" select="$N - 1"/>
       <xsl:value-of select=" $N * kl:fakt($M)"/>
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select="1"/>
      </xsl:otherwise>
    </xsl:choose>

 </xsl:function>

</xsl:stylesheet>
```

15. Mintapélda. Az alábbi összefoglaló példában a bemenő dokumentum a termékek eladási árait tartalmazza. Az igényelt XSLT kódnak ebből az adatsorból egy egyszerű grafikont kell képeznie a HTML nyelv lehetőségeire építve. A bemenő dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xx9.xsl" ?>

<termekek>
 <termek>
   <kod>IUC</kod> <kategoria>CD</kategoria> <ar>1233</ar> <db>23</db>
 </termek>
 <termek>
    <kod>HH1</kod> <kategoria>B</kategoria> <ar>833</ar><db>12</db>
 </termek>
 <termek>
    <kod>OT7</kod> <kategoria>CD</kategoria> <ar>1433</ar><db>12</db>
 </termek>
 <termek>
    <kod>DGW</kod> <kategoria>B</kategoria> <ar>454</ar><db>56</db>
 </termek>
 <termek>
    <kod>JIU</kod> <kategoria>CD</kategoria> <ar>760</ar><db>31</db>
 </termek>

</termekek>
```

A program előbb összesíti a kategóriánkénti darabszám értékeket ('db' elem), majd ehhez egy kis grafikont készít a HTML nyelv 'HR' elemének segítségével. Az oszlopok vízszintesen helyezkednek el, s az igazítást egy 'TABLE' elemen keresztül oldottuk meg. Az elkészült konverziós program:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:kl="kl1.kl1">
 <xsl:output media-type="text/html" omit-xml-declaration="yes"  include-content-type="no"
```

```
><</xsl:output>

  <xsl:variable name="vonals">12</xsl:variable>
  <xsl:variable name="vonalarany">4</xsl:variable>
  <xsl:template match="/">
    <xsl:copy>
      <xsl:element name="html">
      <xsl:element name="head"> <xsl:element name="title">Proba</xsl:element></xsl:element>
      <xsl:element name="body">
        <xsl:apply-templates  select="termekek"></xsl:apply-templates>
      </xsl:element>
      </xsl:element>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="termekek">
    <xsl:element name="table">
     <!--  <xsl:attribute name="align" select="'left'"/> -->
      <xsl:attribute name="border" select="3"/>
      <xsl:element name="thead">
        <xsl:element name="tr">
          <xsl:element name="td"> nev </xsl:element>
          <xsl:element name="td">osszdb</xsl:element>
        </xsl:element>
      </xsl:element>

    <xsl:for-each-group select="termek" group-by="kategoria">
      <xsl:element name="tr">
       <xsl:element name="td">
            <xsl:value-of select="current-grouping-key()"/>
       </xsl:element>
       <xsl:element name="td">
          <xsl:attribute name="align">left</xsl:attribute>
          <xsl:element name="hr">
            <xsl:attribute name="width">
               <xsl:value-of select="sum(db)*$vonalarany"/>
             </xsl:attribute>
            <xsl:attribute name="size">
              <xsl:value-of select="$vonals"/>
             </xsl:attribute>
          </xsl:element>
       </xsl:element>
      </xsl:element>
    </xsl:for-each-group>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

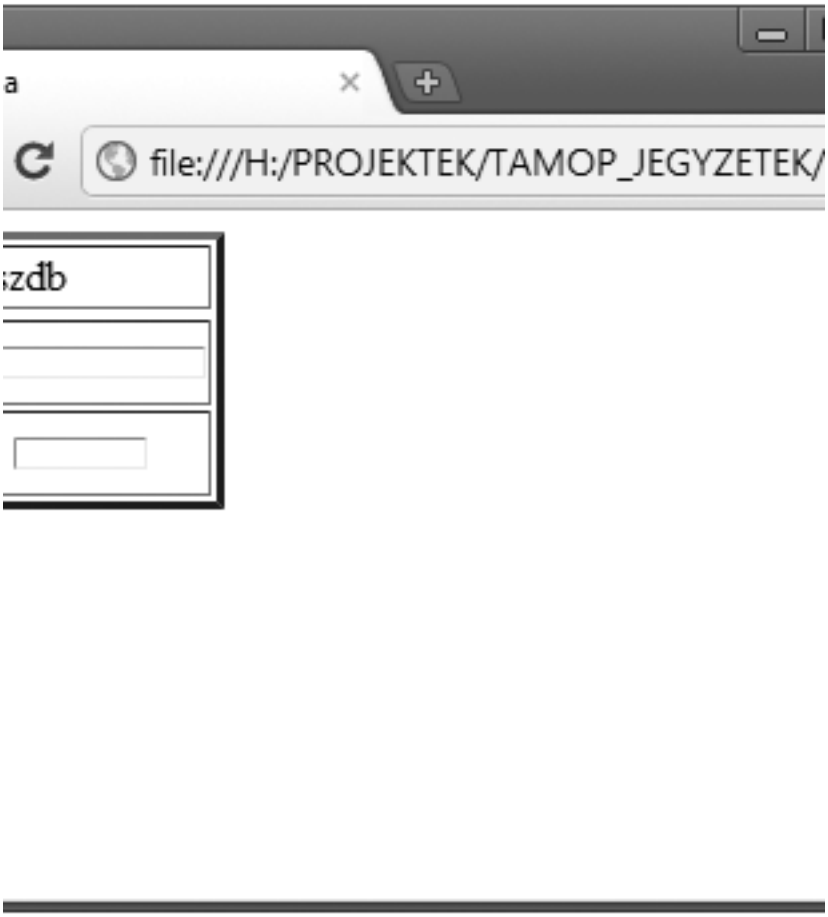
A programban globális változókon keresztül lett megadva a megjelenítéshez kapcsolódó két paraméter, a vonal vastagsága és vonalhossz arányszám. Az program további sajátossága, hogy most a kimenet nem XML állomány, hanem HTML. Az 'output' elemen keresztül lehet megadni a kimeneti dokumentum fejrészét, a dokumentum típusát. Az elkészült kimeneti dokumentum:

```


<html>
  <head>
```

```
    <title>Proba</title>
  </head>
  <body>
    <table border="3">
      <thead>
        <tr>
          <td> nev </td>
          <td>osszdb</td>
        </tr>
      </thead>
      <tr>
        <td>CD</td>
        <td align="left">
          <hr width="92" size="12">
        </td>
      </tr>
      <tr>
        <td>B</td>
        <td align="left">
          <hr width="48" size="12">
        </td>
      </tr>
    </table>
  </body>
</html>
```

A böngészőben megjelenő kép:



15. A generált HTML lap megjelenése

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      [Root:<xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
      <xsl:apply-templates/>
   </xsl:template>


   <xsl:template match="node()">
         [element:<xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
          <xsl:for-each select="namespace::node()">
                prefix: <xsl:value-of select="name(.)"/>
          </xsl:for-each>
     { <br/>
      <xsl:apply-templates/>
      <xsl:apply-templates select="namespace::*"/>
      <xsl:apply-templates select="attribute::*"/>
      }   <br/>
   </xsl:template>


   <xsl:template match="text()">
```

```
      [text: <xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="comment()">
      [comment: <xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="processing-instruction()">
      [proc: <xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="@*">
      [attribute:<xsl:value-of select="local-name()"/>: <xsl:value-of select="."/>] <br/>
      <xsl:apply-templates/>
   </xsl:template>


</xsl:stylesheet>
```

2. fejezet - Az xQuery nyelv áttekintése

1. Az XQuery működési modellje

Az XML szabvány áttekintése során már láthattuk, hogy az XML dokumentum egy fa szerkezetű struktúrával reprezentálható. A reprezentáció nemcsak az adatok tárolását, hanem a feldolgozás menetét is jelentősen befolyásolja. Például a DOM modellben alapvetően a navigációs lépéseken keresztül juthatunk el a vizsgált csomópontokig, s az adatkezelő műveleteknél csomópontonkénti feldolgozást hajtunk végre. Vegyük példaként az alábbi sémát:

```
<xs:element name="adatok" type="adatok_tipus" />

<xs:complexType name="adatok_tipus">
<xs:choice maxOccurs="unbounded">
  <xs:element name="auto" type="auto_tipus">
  <xs:element name="tulaj" type="tulaj_tipus">
</xs:choice>
</xs:complexType>

 <xs:complexType name="auto_tipus">
  <xs:sequence>
    <xs:element name="rendszam" type="xs:string">
    <xs:element name="tipus" type="xs:string">
    <xs:element name="ar" type="xs:integer">
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="tulaj_tipus">
   <xs:sequence>
     <xs:element name="kod" type="xs:string">
     <xs:element name="nev" type="xs:string">
   </xs:sequence>
 </xs:complexType>
```

Ha módosítani szeretnénk a Fiat típusú autók árát az alábbi séma mellett, akkor előbb el kell jutnunk navigációs műveletekkel mindazon csomóponthoz, amely ár mezőt tartalmaz és a Fiat típusú autó leíró csomóponthoz tartozik, majd ott egy újabb paranccsal átírhatjuk a csomóponthoz tartozó szövegértéket. Mivel a logikailag egybetartozó típuselem és az ár elem két különböző elérési útvonalon helyezkedik el, ezért nem lehet egyszerűen leírni a módosítást elvégző tevékenység kódját sem. Ehhez előbb át kell sétálni a fa autó típusú csomópontjain (szerencsére erre van egy egylépcsős utasítás), majd ott előbb megkeressük a típus csomópontot, s ha az megfelel, akkor visszatérve az autóhoz, egy másik útvonalon keresztül lehet elmenni az igényelt ár mezőhöz. Mindezen lépések egy viszonylag összetett vezérlési szerkezetet igényelnek.

Az ilyen procedurális megközelítés hátránya, hogy a nagyobb időbefektetéssel oldható meg a feladat és nagyobb a hibalehetőség, hiszen hosszabb ideig, nagyobb részre kell a programozónak koncentrálnia. A kezelés hatékonyságának növelésére dolgozták ki az imperatív nyelveket, melyekben egy magasabb szintű, átfogó parancson keresztül lehet megadni az elvégzendő műveletet. Az adatbázis kezelés világában az SQL nyelv egy ilyen imperatív nyelv, melyben a fenti példafeladat a megfelelő relációs sémában gondolkozva az alábbi alakot ölti:

```
UPDATE autok SET ar = ar + 100 WHERE tipus='Fiat'
```

Nyilvánvaló, hogy a fenti parancs sokkal jobban áttekinthető, mint a féloldalnyi- többoldalnyi forráskód. Természetesen egy ilyen imperatív nyelv esetén szükség van egy olyan értelmezőre, amely a magas szintű utasítást át tudja alakítani alacsonyabb szintű forráskódra. Az adatbáziskezelőknél az SQL Query Engine végzi el ezt a feladatot. Mivel a fejlesztés szempontjából a gyakoribb feladatok ellátására hasznosabb egy imperatív nyelv, korunk információs rendszerei törekszenek egy ilyen SQL-szerű nyelv beépítésére. Az XML adatkezelésnél is megfigyelhető ilyen törekvés. Az XML adatkezelésnél az XSLT részben egy ilyen jellegű nyelvnek tekinthető, hiszen több alacsonyabb szintű kódrészletet is igényel. Az XSLT szemléletmódja az algoritmus alapú szemléleten nyugszik, hiszen a fejlesztő egy navigációs algoritmust lát maga előtt, melynek szabályozása egy tömör és hatékony módon megadható. Az XSLT nyelv lényeges megszorítása, hogy alapvetően csak a lekérdezésre koncentrál. Az XSLT célja egy megadott XML dokumentumból egy transzformált XML dokumentum előállítása. Ez az irány abból fakad, hogy az XML dokumentumokat alapvetően csak bemenő, leíró adatként használják, amelynek tartalma egy másik, dinamikus adatkezelést biztosító adatkezelő rendszerből származik. Ekkor a változtatásokat a forrás adatrendszeren keresztül végezhetjük el. Ha most az egyszerűség kedvéért, egy lekérdezésre alakítjuk a mintafeladatot, akkor a végrehajtandó művelet szövege: Adja meg a Fiat típusú autók ár értékeit. A kapcsolódó XSLT utasítás (egy lehetséges megvalósítás):

```
<xsl:template match="/">
  <xsl:apply-templates select="/auto"/>
</xsl:template>
<xsl:template match="auto">
 <xs:if test="tipus/text()='Fiat'">
    <xsl:value-of select="ar"/>
 </xs:if>
</xsl:template>
```

A mintához két további megjegyzést fűzünk hozzá. Egyrészt a módosítást is meg lehet oldani kerülő úton, úgy hogy az alap XML dokumentumból a nem érintett ár mezők kivételével minden más csomópontot átviszünk az eredménybe változtatás nélkül és az érintett ár csomópontoknál az új árértéket tartalmazó szövegcsomópontot tesszük be gyerekként. Ezen lekérdezés megvalósítása igényli a séma pontos ismeretét, hogy kijelölhessük a kódban mely csomópontoknál lehet a teljes részfa átvitelét, másolását megoldani. Második kiegészítés, hogy azért az XML világból sem maradt ki teljesen a módosítás jellegű műveletek kezelése. A később ismertetendő XML adatbázisoknál a tevékenységek egyik fontos eleme az adatkezelő utasítások biztosítása és az ezen fejezetben ismertetendő XQuery nyelvben is a közeljövőben kidolgozandó funkciók közé tartozik az adatkezelés megvalósítása.

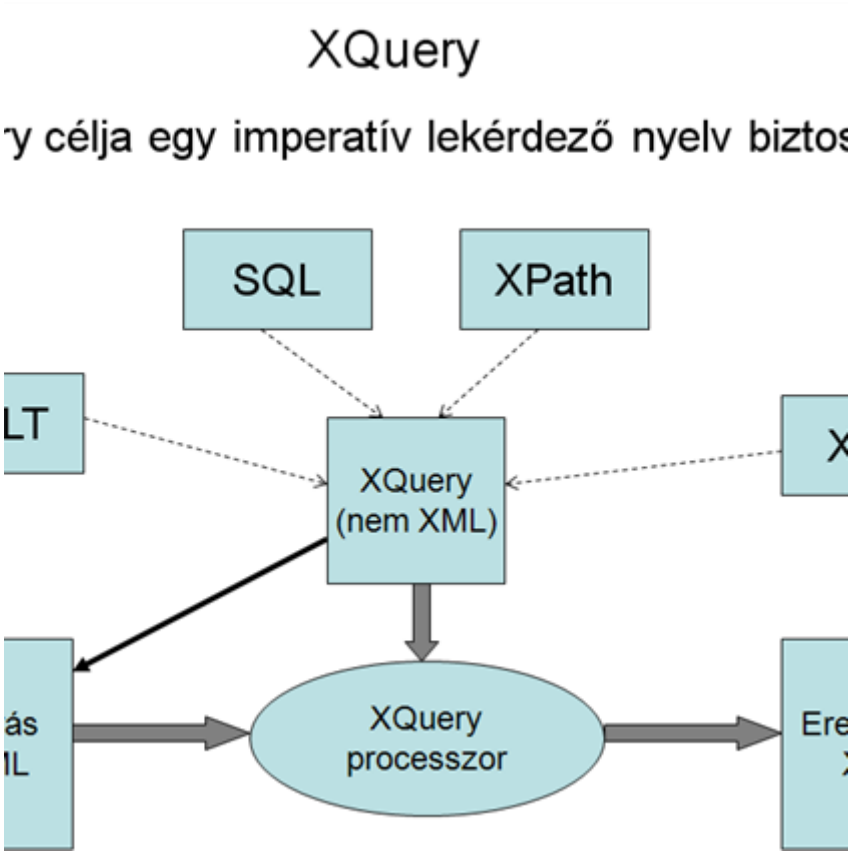
Most azonban még a lekérdezéseknél maradva, egy új megközelítésre térünk rá, az XQuery-re. Az XQuery szabvány bizonyos értelemben az XSLT vetélytársának tekinthető, mivel ez is az XML dokumentumok hatékony lekérdezését szolgálja. Egy lényeges különbség az XSLT-vel szemben, hogy itt erősebb az SQL imperatív jellege, s a működési és használati környezetet is az SQL-re emlékeztet. A parancsok szintaktikája viszont egy átmenetet jelent az XSLT rekordonkénti és az SQL halmazorientált szemlélete között. Az XQuery projekt valamivel az XSLT projekt beindulása után kezdődött el, és fő céljának az XML adatlekérdezéshez egy magas szintű lekérdező felület biztosítása tekinthető.

Az XQuery nyelv, mint W3C szabvány 2007-ben jelent meg. Sokáig, még a tervezés fázisában, megkérdőjelezték szükségességét, hiszen az XSLT is hasonló funkciót tölt be, és az XSLT érett, már elterjedt volt az XQuery megjelenésekor. Az ellenzők tábora mellett ott van a pártolók tábora is, hiszen az XSLT és XQuery két eltérő funkciót szolgál. Az XSLT alacsony szintű, rugalmas XML transzformáció leíró, míg az XQuery egy magas szintű lekérdező nyelv. A XQuery mellett felhozható érvek közül ez egyik legfontosabb az egyszerű formalizmus mellett az a tény, hogy a W3C terveiben az XQuery már nemcsak lekérdező nyelv, hanem adatkezelő nyelv is (XQuery Update Facility), mely funkció teljes egészében hiányzik az XSLT nyelvből.

Az XQuery legfontosabb, átfogó tulajdonságai a következőkben foglalhatók össze:

- Halmazorientáltság: az XQuery kifejezésekben lehetőség van arra, hogy egy szimbólummal egy csomóponthalmazt reprezentáljunk, s a műveletet (mint például egy szelekciót) ezen halmazra értelmezzünk;
- Gazdag kifejezőerő: az XQuery operátorai, mint egy algebra operátorai, tetszőleges egymásba ágyazást tesznek lehetővé, itt is él az egymásba ágyazott lekérdezések lehetősége;
- XML bemeneti adat és XML kimeneti adat: Az XQuery, az XSLT-hez hasonlóan XML állományokon értelmezett és az eredmény is XML formátumú. Egy lényeges eltérés az XSLT-hez képest, hogy az XQuery parancs nem a forrás XML dokumentumba épített, mint az XSLT, ahol a forrás XML dokumentumba van megadva az XSLT transzformáció meghívása. Az Xquery esetében egy külön állományban van az XQuery parancs, amelyben hivatkoznunk kell a forrás állományokra;
- A parancsok nem XML formátumban adottak: Egy lényeges eltérés a korábban megismert XML kezelő felületekkel szemben, hogy itt nem törekedtek az XML formátum következetes keresztülvitelére. Az XQuery nyelv elemei alapvetően nem XML szabványúak, habár létezik a szabványnak egy XML alapú változata is, az XQueryX változat. Az XML formátumtól való eltérés oka, hogy ez által lényegre törőbb, áttekinthetőbb lett a parancsnyelv;
- XPath alapú: A XQuery nyelv is a XML dokumentumok fa reprezentációját használja. Az egyes csomóponthalmazok kiválasztása az XPath szabványon alapszik, amely halmazorientált megközelítést biztosít. A XQuery az XPath funkciót további műveletekkel egészíti ki.
- SQL-hez hasonló szemlélet: A XQuery rokonsága az SQL nyelvvel a tervezési szempontokban és a kezelési módszertanban is megnyilvánul. A közös vonások közé tartozik a halmazszemlélet, az ad-hoc jelleg, a magas szintű imperatív nyelv és a rugalmas egymásba ágyazhatóság;
- Procedurális elemeket is tartalmaz: A funkcionalitás kiterjesztése céljából lehetőség van egyedi procedurális elemek beépítésére is, tehát létre lehet hozni saját eljárásokat és függvényeket. Emellett az alap lekérdező nyelvi is kettősséget mutat ezen a téren is, hiszen a halmazorientált elemek mellett egyedi, csomópont szintű feldolgozási utasítások is vannak, amikor csak az egyes szimbólumok csak egyetlen csomópontot reprezentálnak.

Nézzük meg, hogyan valósulnak meg az említett tulajdonságok, milyen működési struktúra tartozik az XQuery rendszerhez. Az XQuery rendszer működésének egyszerűsített vázlatát mutatja be az alábbi ábra.



16. Az xQuery rendszer működési vázlata

A modellből kivehető, hogy egy külön állományt kell készíteni az XQuery parancs leírására. A parancsban szerepelnek a forrás XML állományok elérési adatai is. A XQuery parancs feldolgozására több korábbi szabvány is hatással van. Egyrészt az XML dokumentumok fa struktúra modellje, az XDM, hiszen a műveletek operandusai a facsomópontok lesznek. A csomópontok kiválasztásánál az XPath szabvány játszik szerepet. A lekérdezési operátorok jellegében, kapcsolódásában az SQL szabvány hatása érvényesül. A lekérdezési műveletek alacsony szinten az XSLT -hez hasonló alakra konvertálódnak. Az lekérdezésben megadható, hogy mely cél XML állományt hozza létre az eredmény dokumentum tárolására.

Az XQuery formátumának szemléltetésére egy kis mintát veszünk. Legyen adott az alábbi forrás XML dokumentum, melynek azonosító neve xx9.xml.

```
<adatbazis>
 <autok>
  <auto rsz="r1">
    <tipus> Opel</tipus>
    <ar>214 </ar>
  </auto>
  <auto rsz="r2">
    <tipus> Fiat</tipus>
    <ar>165 </ar>
  </auto>
  <auto rsz="r4">
    <tipus> Skoda</tipus>
    <ar>365 </ar>
  </auto>
  ...
 </autok>
 <emberek>
  <ember kod="k1">
```

```
      <nev>Zoli</nev>
      <varos>Eger</varos>
    </ember>
    ...
  </emberek>
</adatbazis>
```

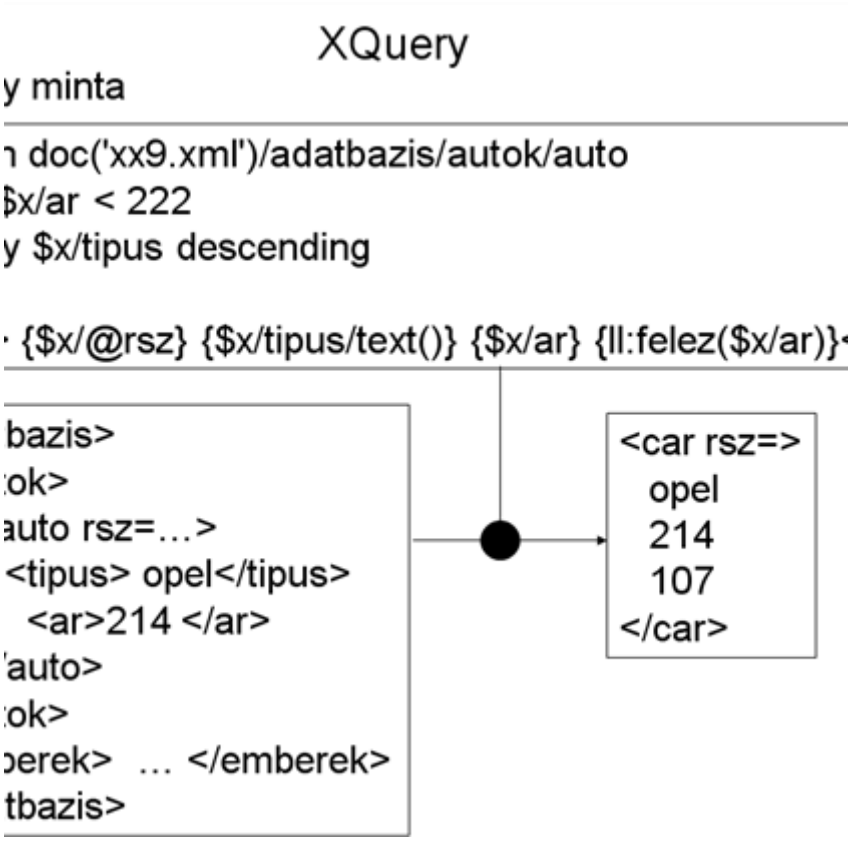
A lekérdezési mintafeladat a 250-nél olcsóbb autók rendszámának és típusának kiíratása. Az XQuery rendszerben a lekérdezést egy külön dokumentumban kell megadni. A lekérdezés alakja:

```
for $x in doc('xx9.xml')/adatbazis/autok/auto
where $x/ar < 250
order by $x/tipus descending
return
  <car> {$x/@rsz} {$x/tipus/text()}</car>
```

Első pillanatra feltűnik, hogy itt nem XML formátum szerepel. A lekérdezés első sorában megadjuk, hogy az xx9.xml dokumentumból emeljük ki a megadott elérési útvonallal (/adatbazis/autok/auto) rendelkező csomópontokat. A kiemelt csomópontok halmazában sorra veszi a feldolgozó azon csomópontokat, ahol az ár mező kisebb, mint 250. A kapott csomópontokat típus szerint sorba rendezi, majd minden csomópontra előállít egy eredménydokumentumbeli csomópontot. A létrejött csomópont tagneve 'car' és az autó rendszám elemtulajdonságát és típus elemértékét tartalmazza. A létrejött XML dokumentum tartalma:

```
<car rsz="r2">
  Fiat
</car>
<car rsz="r1">
  Opel
</car>
```

Mint a példa is mutatja, az XSLT-hez hasonlóan itt sem garantált, hogy az eredmény dokumentum helyesen formált XML dokumentum lesz. A lekérdezést úgy kell formálni, hogy a kívánt szerkezetű válasz XML dokumentumot szolgáltassa.



17. xQuery forrás és cél állományok szerkezete

Az XQuery lekérdezés feldolgozó motorja több lépcsőfokon keresztül jut el a kért eredményig. A következőkben ezen feldolgozási fázisokat tekintjük át. A feldolgozás három fő fázisból áll:

- előfeldolgozás,
- statikus elemzés,
- dinamikus elemzés.

Az elő feldolgozás során a bemeneti XML dokumentumokból, melyek szöveges alakban érkeznek be, létrejön a dokumentumok fa modellje. Az átalakítás fontosabb lépései:

1. Az XML szöveges, linearizált alakjának elemzése (parsing). Az elemzés során egy SAX jellegű feldolgozásnak megfelelően meghatározásra kerülnek az egyes elemek határai és alkotó gyerekelemei.
2. A felárt elemekből egy Infoset modell létrehozása. Mint korábban már vettük, az Infoset modell egy logikai szintű leírása a fában található csomópontoknak.
3. A kialakított Infoset modell validálása. Ehhez a dokumentumhoz csatolt XMLSchema leírást dolgozza fel az elemző és összeveti az XML dokumentumot a kapcsolt sémaleírással.
4. Az ellenőrzés eredményeképpen kapott Infoset elemekből felépíti az érvénye információs elemkészletet (PSVI: post-shema validation infoset).
5. A dokumentum fa modelljének előállítása. A fa leírására itt is a jól ismert XDM adatmodellt használják fel a feldolgozók. Ebben a leírásban a forrásból vett közvetlen adatok mellett számos kísérő információ is fellelhető, mint például a kapcsolt adattípus megadása a szövegértékhez.

Az elő feldolgozás során létrejött dokumentumfára fogják értelmezni az XQuery lekérdezést. A lekérdezés feldolgozása, az SQL feldolgozáshoz hasonlóan a parancs alakjának szintaktikai elemzésével kezdődik. A statikus elemzési fázis egy normalizált műveleti gráf előállításával zárul. A fázis lépései:

1. Az XQuery parancs szintaktikai elemzése, a parancs felbontása tokenekre, kisebb nyelvtani egységekre. A felbontás során meghatározásra kerülnek, hogy mely parancsszavak és ezáltal mely műveletek kerülnek meghívásra.

2. Az előző lépésben előállított művelethalmazból felépítésre kerül egy műveleti gráf. A műveleti gráf irányított élű gráf, a csomópontjaiban állnak a műveletek, a levelekben az induló operandusok szerepelnek és akkor mutat egy A csomópontból a B csomópontba, ha a B felhasználja az A művelet eredményét. A műveleti gráf optimalizálásával biztosított a hatékony válaszidő.

3. A feldolgozás ezen lépésében a műveleti gráf kiértékeléséhez szükséges információk összegyűjtése történik meg. Ennek során a működési kontextus adatait gyűjti össze a rendszer, meghatározva ez egyes rendszerparaméterek, névtérparaméterek, állományjellemzők értékeit.

4. A begyűjtött paraméterekből egy kontextus-leíró struktúra jön létre, melyet a feldolgozás további fázisában, a dinamikus kiértékelésnél használ fel a feldolgozó.

5. Normalizált műveleti gráf előállítása. A normalizálás során egyes implicit paramétereket explicit alakra hoz a rendszer. Ilyen implicit érték lehet például a névtér, melynek a default névtér alakja explicit névtér kijelölésre fog változni.

A feldolgozás harmadik fázisa a dinamikus kiértékelés. Ennek során az előállított műveleti gráf, a létrejött XML dokumentum modell és a kontextusleírók alapján ez eredmény dokumentumot hozzák létre. A fontosabb lépések a fázison belül:

1. A műveleti gráf átalakítása alacsonyabb szintű algoritmus fává. Az egyes magasabb szintű műveletekhez kiválasztásra kerül az implementációs algoritmus.

2. A kifejezések kiértékelése. A műveletek végrehajtása során a leírásban szereplő kifejezések egy-egy konkrét értékkel helyettesítődnek. A kiértékelés során kerülhet meghatározásra az egyes kifejezések aktuális, dinamikus adattípusa is.

3. A kiértékelés során előállnak a válasz dokumentum alkotó elemei, melyeket feldolgozónál kimeneti atomoknak neveznek. Ezek egy-egy Infoset elemnek felelnek meg.

4. A műveleti fa feldolgozás végén előáll minden igényelt Infoset elem. A felhasználó természetesen nem ezt a logikai leírást várja el. Ezért első lépésében az Infoset készletből egy XDM fa modell épül fel.

5. A végleges, kimeneti eredmény előállításhoz az XDM fa modellt még sorosítani, linearizálni kell, melynek végén előáll az igényelt szöveges XML dokumentum formátum.

Az elemzés során a műveletek meghatározása mellett az egyik fontos tevékenység a kifejezések konzisztencia vizsgálata. A konzisztencia vizsgálat során ellenőrzésre kerülnek többek között a hivatkozások és adattípusok. Az elemzés során több különböző hibatípus is felléphet. A következő lista néhány tipikus hibatípust ad meg az ellenőrzés jellegének bemutatására:

- statikus szintaktikai hiba (például a FOR kulcsszó helyett a FRO szó szerepel a parancsban),
- statikus típushiba (például egy 'name()' kifejést egy csomópont típusú kifejezéssel hasonlítunk össze),
- dinamikus értékhiba (például nullával való osztás, túlcsordulás),
- dinamikus típushiba (például egy text() aktuális értéke nem konvertálható egész számra).

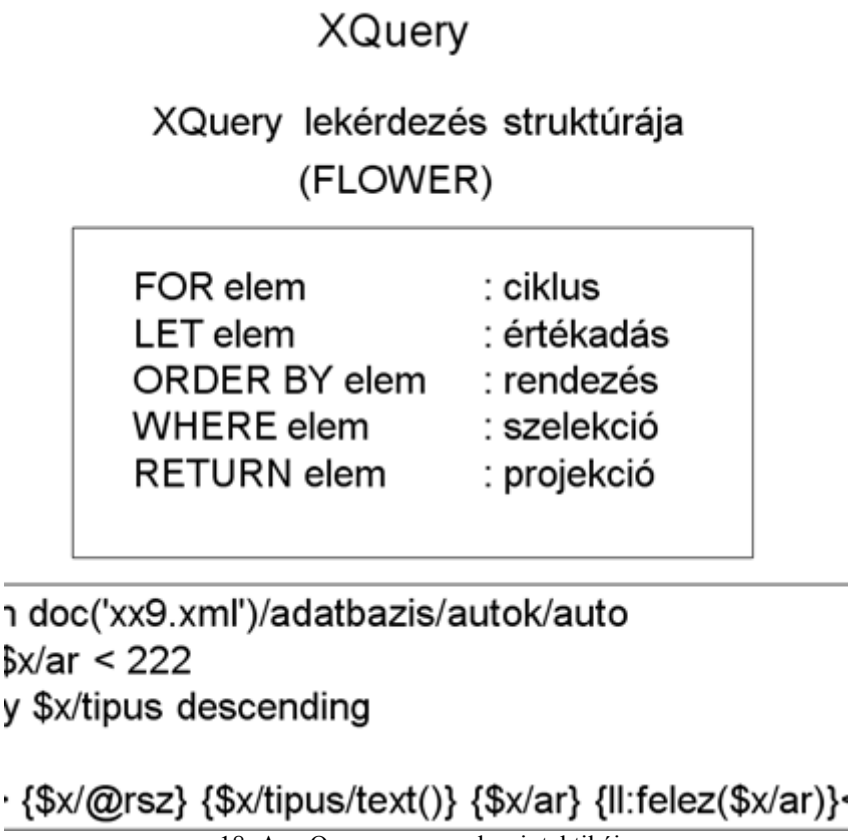
2. Az XQuery lekérdezési parancsai

Az XQuery lekérdés formalizmusa az alábbi logikára épül. Elsőként kijelöljük, hogy mely csomóponthalmazt vagy csomópontot kívánjuk feldolgozni, hogy abból információt merítsünk. A kijelölés során egy-egy változót rendelhetünk a halmazhoz vagy csomóponthoz. Halmaz esetén a megadott változó végigfut a halmaz értékein és minden értéknél elvégzi a magban megadott műveleteket. A kijelölés során megadható egy szelekciós kifejezés, amely leszűkíti a feldolgozott csomópontok körét. A halmazhoz még egy feldolgozási sorrendiséget kijelölő művelet is tartozhat. A magban a megadjuk, hogy milyen kimeneti atomokat állítson elő az éppen feldolgozás

alatt álló csomóponthoz. A feldolgozás magjában egy újabb lekérdező kifejezést lehet beágyazni. A lekérdezés általános formátuma:

```
FOR $v IN kif1
LET $w := kif2
WHERE kif3
ORDER BY kif4
RETURN kif5
```

A FOR kulcsszó mögötti részben lehet megadni a csomóponthalmazt. A \$v szimbólum egy változót jelöl, a változók neve előtt mindig szerepel a dollárjel. A kif1 kifejezés egy csomóponthalmazt takar, melyben XQuery és XPath elemekre építve adjuk meg a halmaz definiálását. A LET kulcsszó mögött olyan változó áll, mely egyetlen értéket vesz fel, e mögött nincs érték iteráció. A WHERE tag a szelekciót, a csomópont szűrést jelöli ki. Csak azon csomópontok kerülnek át az eredménybe, melyekre teljesül a megadott kif3 logikai kifejezés. A RETURN részben kell megadni az eredmény dokumentum felépítését. Az egyes parancstagok kezdőbetűiből felépítetten a fenti lekérdezés alakot FLOWER-kifejezésnek is nevezik.



18. Az xQuery parancsok szintaktikája

A forrás csomópontok kijelölésé többféleképpen történhet az alábbi módokon:

- Konstanssal, literállal. Ekkor a kifejezésben felsoroljuk a lista elemeit. Példaként egy háromelemű listát adunk meg:

```
for $x in ('a','b','s')
```

- XML forrás dokumentum használunk. Ehhez az állomány elérési útvonalát kell megadni, az elérési útvonalat egy fn:doc() függvény argumentumában kell szerepeltetni. A dokumentum kijelölés után szerepelhet egy XPath kifejezés is, mely a dokumentumon belül szűkíti le az érintett csomópontok körét.

```
for $x in fn:doc("xx9.xml")/adatbazis/autok/auto
```

- XML csomópont gyűjtemény megadása. Ekkor egy saját, implementáció függő módon lehet a csomópontokat összeszedni egy megadott URI értékhez. A csomópont gyűjteményt az fn:collection() függvényen keresztül érhetjük el.

```
for $x in fn:collection("http:://a.b")
```

A csomópont kijelölő kifejezésekben a dokumentum részeinek kijelölésénél több gyári illesztési függvényt is használhatunk. Ezek segítségével ellenőrizhetjük a csomópontok típusát, jellegét és előfordulási számosságát is. A következő lista ezen elemekből mutat be néhány reprezentánst.

- element() : bármely csomópont,
- element(A,B) : A nevu, B típusú csomópont,
- attribute(A,B) : A nevu, B típusú elemjellemző,
- text() : szövegcsomópont,
- node() : bármely csomópont,
- node()* : csomópontok listája, tetszőleges sok elemmel,
- attribute() + : egy vagy több elemjellemző,
- atribute(rsz) : rsz nevű elemjellemző,
- element(*,B)? : opcionális B típusú elem,
- derives-from(A,B) : A típus a B-ből származik-e (közvetlenül nem hívható meg),
- item() : csomópont vagy érték,
- comment() : megjegyzés,
- node()+ : csomópontok listája, minimum egy elemmel.

Az előző példában is szereplő literál lista megadása kétféle módon is történhet. Egyrészt felsorolással, másrészt intervallum kijelöléssel:

```
$x in (1,2,4,5,6)
$x in 1 to 6
```

Egy érdekes kérdés a relációs operátorok használata, amikor is két értéket, két operandust összehasonlítunk, hogy eldöntsük, relációban állnak-e egymással vagy sem. Az összehasonlítás érdekessége, hogy két csomópont esetén, mivel a csomópontok összetett szerkezetek különbözőképpen lehet értelmezni az egyezőséget. Lehet a teljes egyezésre is gondolni és lehet csak szövegelemekre történő egyezést is kérni. Emiatt különböző relációs operátorok élnek az egyezőség vizsgálatra:

- skalár érték összehasonlítás:
  - eq: egyenlő,
  - ne: nem egyenlő,
  - lt: kisebb mint,
  - le : kisebb egyenlő,
  - gt : nagyobb mint,

    - ge :nagyobb egyenlő.
- szekvencia, tartomány összehasonlítás:
    - = : van két azonos elem a két oldalról,
    - < : van két kisebb relációban álló elem a két oldalról,
    - > : van két nagyobb relációban álló elem a két oldalról,
    - != :: van két nem egyenlő elem a két oldalról.
- csomópont összehasonlítás:
    - is : azonos csomópontok,
    - << : megelőzési reláció a dokumentum sorrend alapján,
    - >,> : megelőzési reláció a dokumentum sorrend alapján.

Az érték összehasonlításnál csak a csomópontok értéke számít. Így például a

```
<a>3</a> eq <b>3</b>
```

kifejezés igaz értékű lesz, mivel a szöveges tartalomérték mindkét operandusnál a 3-as érték. A halmazműveleteknél akkor teljesül a reláció, ha van olyan elempár a két oldalról, amelyre teljesül a feltétel. Például a

```
(1,2) = (2,3)
```

és az

```
(1,2) != (2,3)
```

kifejezés is igaz értékű, hiszen mindkettőre van példa a listákban.

Az eredménycsomópont felépítésénél szerepelhetnek statikus és dinamikus elemek is. A statikus elemeknél a parancs szövegében szereplő elemek kerülnek át közvetlenül. Statikus elem lehet csomópont leírás vagy szöveg leírás. Erre mutat példát az alábbi minta:

```
return
<auto>
   <tipus> Fiat </tipus><ar>234</ar>
</auto>
```

A statikus adatok mellett a dinamikus elemek teszik rugalmassá a rendszer működését. A dinamikus elemeknél egy kifejezés kiértékelési értéke fog átkerülni az eredmény dokumentumba. Az XQuery szabvány megengedi, hogy mind a csomópont séma jellemzők, mind a szöveges tartalom dinamikusan generálódjon. A dinamikus kifejezés kiértékelésének operátora a kapcsos zárójelpár:

```
{ kifejezés }
```

A fenti művelet hatására a kifejezés értéke kerül az eredménybe, mint azt az alábbi példa is mutatja. Az

```
RETURN
  <ar> {234 + 123} </ar>
```

utasítás hatására az eredmény csomópont alakja:

```
<ar> 357 </ar>
```

Az XQuery nyelv egyes utasításainak részletes bemutatásához egy minta XML állományt veszünk (minta.xml), melynek tartalma a következő: >

```
<adatbazis>
  <autok>
    <auto rsz="r1" tulaj="1">
      <tipus> Opel</tipus>
<ar>214 </ar>
<szin>kek</szin>
    </auto>
    <auto rsz="r2" tulaj="2">
      <tipus> Fiat</tipus>
<ar>165 </ar>
<szin>piros</szin>
    </auto>
    <auto rsz="r4" tulaj="1">
      <tipus> Skoda</tipus>
<ar>365 </ar>
<szin>feher</szin>
    </auto>
  </autok>
  <emberek>
    <ember kod ="1">
<nev>Peter</nev>
<varos>Miskolc</varos>
    </ember>
    <ember kod="3">
<nev>Anna</nev>
<varos>Gyongyos</varos>
    </ember>
    <ember kod ="2">
<nev>Zoli</nev>
<varos>Miskolc</varos>
    </ember>
  </emberek>
</adatbazis>
```

2.1. Egyetlen csomópont feldolgozása

Ha forrás dokumentumból, köztes csomóponthalmazból csak egyetlen csomópontra vagy értékre van szükségünk, akkor használhatjuk a

```
let $valtozo := elem_kijelölés
```

utasítást, s az ezt követő utasításrészekben a \$változó szimbólum mögött a kijelölt elem fog szerepelni. Például, az

```
let $x := ('a','b','s')
  return <a> {$x} </a>
```

hatására egyetlen <a> elem kerül kiírásra, melynek tartalma a megadott elemhármas:

```
<a>
  a
  b
  s
</a>
```

Az autók adatainak kiíratására szolgáló utasítás alakja:

```
let $a := fn:doc("minta.xml")/adatbazis/autok
  return
  <lista> {$a} </lista>
```

2.2. Több csomópont feldolgozása

Ha egy csomóponthalmaz, elemhalmaz elemeivel kell dolgozni, akkor egy iterációs ciklust kell kijelölni a

```
for $valt in elem_halmaz
```

utasítással. Ekkor a $valt változó sorba felveszi a halmazbeli értékeket. Például a

```
for $v in ('a','b','c')
return
  <elem> {$v} </elem>
```

utasítás hatására három 'elem' csomópont fog létrejönni:

```
<elem> a </elem>
 <elem> b </elem>
 <elem> c </elem>
```

A következő példában az autók típusát fogjuk kiíratni:

```
for $a in fn:doc("minta.xml")\\auto
return
   <typ> {$a/text()} </typ>
```

Vegyük észre, hogy az eredmény megadásánál nem a

```
return
  <typ> {$a} </typ>
```

utasítást adtuk ki, mert a \$a szimbólum a teljes csomópontot jelöli, s ebben az esetben a typ csomópont alatt egy teljes auto csomópont megjelenne. A

```
return
  <typ> {$a/text()} </typ>
```

esetben viszont a text() XPath függvénnyel a csomópont szövegtartalmát emeljük ki és tesszük át az eredménybe, tehát nem fog megjelenni külön gyerekelem-csomópont.

2.3. Iterációk, értékadások kapcsolása

Egy XQuery utasításban több let és for utasításrész is szerepelhet. Ekkor a belső iteráció a külső iteráció minden egyes értékére lefut, tehát vezérlési szerkezet megközelítésből ekkor egy beágyazott ciklus szerkezetet kapunk. Ez a mechanizmus alkalmas például a Descartes-szorzat megvalósítására, amelyet a következő példa is szemléltet:

```
for $x in doc('minta.xml')/adatbazis/autok/auto
for $y in doc('minta.xml')/adatbazis/emberek/ember
return
   <a>  {$x} {$y}  </a>
```

A példában az összes autó-ember csomópontpáros meg fog jelenni az eredményben. Összesen 9 'a' elem jön létre, és mindegyik két gyerekelemmel rendelkezik.

A belső ciklusban felhasználhatók a külső ciklusban definiált változók, azaz a belső rész mindig a külső rész további szűkítésének tekinthető. Az előző példát átalakítva úgy, hogy a külső változót a belső értékadásnál felhasználjuk, az alábbi parancsot kapjuk:

```
let $b := doc('minta.xml')/adatbazis
for $x in $b/autok/auto
for $y in $b/emberek/ember
return
  <a>  {$x}  {$y}  </a>
```

Az eredmény struktúra előállításakor figyelni kell arra, hogy az átvitelkor az elemnek a forrásban betöltött szerepe is átadásra kerül, azaz nemcsak az érték megy át. Így például a

```
let $b := doc('minta.xml')/adatbazis
for $x in $b/autok/auto
for $y in $b/emberek/ember
return
   <a>
     <car>{$x/@rsz}{$x/tipus/text()}</car>
     <owner>{$y/nev/text()}</owner>
   </a>
```

parancsban az 'rsz' csomópont elemjellemzőként fog bekerülni a 'car' csomópont alá, mivel a forrásban is elemjellemző szerepet töltött be.

2.4. Szelekció elvégzése

A feldolgozott elemekből való válogatást, szűrést kétféle módon is megoldhatjuk. Egyrészt a kijelölő XPath kifejezésbe építünk be szelekciós részt, vagy az XQuery FLOWER parancsban szerepeltetünk egy WHERE feltétel részt. Ekkor csak a feltételnek eleget tévő elemek kerülnek feldolgozásra a parancs belsejében. A következő példában a 222-nél drágább autók típusát és rendszámát íratjuk ki. A rendszám továbbra is elemjellemző marad.

```
for $x in doc('minta.xml')/adatbazis/autok/auto
where $x/ar > 222
return
   <car>{$x/@rsz}{$x/tipus/text()}</car>
```

A fenti mechanizmusra építve már szelekciós join műveletet is végre tudunk hajtani. Feladatként az összetartozó autó-ember párosokat kérdezzük le:

```
for $a in fn:doc('minta.xml')//auto
for $e in fn:doc('minta.xml')//ember
where $a/@tulaj eq $e/@kod
```

```
return
  <a>
    <car>{$x/@rsz}{$x/tipus/text()}</car>
    <owner>{$y/nev/text()}</owner>
  </a>
```

A szelekciós feltételben skalárérték-alapú összehasonlítást végeztünk.

2.5. Elemek rendezése

Iteráció esetén lényeges lehet a halmazelemek megjelenési sorrendje is. A rendezés

```
order by kifejezés mód
```

utasításával megadhatjuk, hogy a halmaz elemeit mely kifejezés szerint rendezze, s a móddal rendezés növekvő (ascending) vagy csökkenő (descending) jellegét lehet beállítani. A példában a 222-nél olcsóbb autók adatait íratjuk ki típusnév szerint csökkenő sorrendben:

```
for $x in doc('minta.xml')/adatbazis/autok/auto
where $x/ar < 222
order by $x/tipus descending
return
   <car> {$x/@rsz} {$x/tipus/text()} </car>
```

Mivel az XQuery környezetben a sorrendiség alapfogalom, az order by parancsot a belső, beágyazott parancsokban is lehet használni.

2.6. Csomópontok létrehozása

A dinamikus lehetőségek között szerepel az eredmény szerkezet dinamikus kialakítása is. Lehetőség van arra, hogy az eredmény dokumentum csomópontjait létrehozó parancsokat használjunk, melyben a csomópont paraméterei, mint például az azonosító neve, kifejezésként adható meg. Az XQuery rendszer legfontosabb csomópont létrehozó utasításai:

```
return
  {element {nev} {ertek}}
    {attribute {nev}{ertek}}
    {text {ertek}}
    {processing-instruction {nev} {ertek}}
    {comment {szöveg}
  }
```

Az element utasítás elemcsomópontot generál, az attribute elemjellemzőt, még a text egy szövegcsomópontot hoz létre. A feldolgozó utasítás csomópontját a processing-instruction parancs, míg a megjegyzés csomópontét a comment parancs váltja ki. A parancsokban mind az azonosító név, mind a tartalom dinamikusan adható meg. Az előző autó kilistázó parancsot most dinamikus elemkijelöléssel oldjuk meg:

```
for $x in doc('minta.xml')/adatbazis/autok/auto
where $x/ar > 222
return
  {element  car {text{$x/@rsz}}}
```

A következő példában a kapcsolódó autók és emberek adatait is dinamikus szerkezet felépítéssel adjuk meg:

```
for $a in fn:doc('minta.xml')//auto
for $e in fn:doc('minta.xml')//ember
where $a/@tulaj eq $e/@kod
return
  element eredmeny {
    element auto {$a},
    element tulaj  {$e}
 }
```

Ezt a dinamikus csomópont felépítést használhatjuk ki arra, hogy a forrás elemjellemzőjéből az eredmény elemébe vigyünk át adatot. Az alábbi példa a rendszámot viszi át elemjellemzőből szövegcsomópontba.

```
for $a in fn:doc('minta.xml')//auto
return
 element auto {text{$a/@rsz}}
```

Egy még nagyobb léptékű átrendezést ad meg a következő feladat: Irassuk ki az autókat úgy, hogy az elem neve a rendszám érték legyen, szövegtartalma a típusérték, és az ár érték elemjellemzőbe megy át. Az átalakítás parancsa:

```
for $a in fn:doc('minta.xml')//auto
return
  element {$a/@rsz} {attribute price {$a/ar} , text{$a/tipus}}
```

Mint már azt a bevezetőben említettük, az XQuery rendszer nem fogja automatikusan biztosítani a helyesen formáltságot az eredmény dokumentumoknál. A korábbi példáink egyik jelentős hibája, hogy nincs gyökere a dokumentumnak. Ha egy olyan eredményt szeretnénk előállítani, melyben létezik ilyen gyökérelem, akkor azt a lekérdezésbe is be kell építeni. Mivel a gyökérelem magába foglalja a többi elemet, a lekérdezést leíró parancsot ebbe a gyökérelembe foglaljuk be az alábbi módon:

```
<gyökér>
  FLOWER lekérdezés
</gyökér>
```

Példaként írassuk ki az autók típusait úgy, hogy a kapott dokumentum tartalmazzon gyökér elemet:

```
<adatok>
{
  for $a in fn:doc('minta.xml')//auto
  return
    element car {$a/tipus}
}
</adatok>
```

A példában a gyökérelem elnevezése 'adatok'.

2.7. Feltételes parancsvégrehajtás

Ha a kimenti állomány egyes részei feltételtől függően változnak, akkor a feltételes kiértékelést megvalósító paranccsal építjük fel a kimeneti dokumentumot. A parancs formátuma:

```
return
{
   if (kifejezes1)
   then kifejezes2
   else kifejezes3
}
```

A kifejezés1 igaz értéke esetén az eredmény a kiefejezés2 helyettesítési értéke lesz, míg hamis érték esetén a kifejezés3 értékét kapjuk vissza. A következő példában a számértékkel megadott ár értéket egy szöveges leírással egészítjük ki, megadva, hogy az árat kevésnek vagy soknak találjuk-e:

```
for $x in doc('minta.xml')/adatbazis/autok/auto
return
  <car>
   {$x/@rsz} {$x/tipus/text()} {$x/ar}
   {if ($x/ar>151) then 'sok' else 'keves'}
  </car>
```

Második kapcsolódó példánkban az ár értékét kell forintra konvertálni, ha az dollárban van megadva:

```
<adatok>
{
for $a in fn:doc('minta.xml')//auto
return
   element car { text {$a/tipus},
   element ar  {
      if  ($a/ar/@valuta eq 'USD')
      then  text {250*$a/ar} else  text {$a/ar}}
   }
```

```
  </adatok>
```

A többszörös elágazást csak az if szerkezetek egymásba ágyazásával lehet megoldani:

```
  if (feltétel1a)
  then feltétel2a
  else if (feltétel1b)
    then fetétel2b
    else ...
```

2.8. Aggregációs függvények

A lekérdezések egyik tipikus köre, amikor összesítő értékeket kell előállítani a részletező adatokból. A XQuery egyik hiányossága, hogy nincs ilyen közvetlen csoportképző operátor, mint a GROUP BY parancs az SQL nyelvben. Itt csak kerülő úton lehet megoldani a csoportosítás problémáját. A csoportosításhoz egy

```
  distinct-values()
```

függvény használható, amely az argumentumában megadott értékhalmazból egy olyan értékhalmazt állít elő, melyben nem fordul elő elemismétlődés. Példaként írassuk ki az emberek városait, úgy hogy ne forduljon elő benne ismétlődés:

```
x
  <varosok>
  {
    for $v in fn:distinct-values( fn:doc('minta.xml')//ember/varos )
    return
      <city>{$v}</city>
  }
  </varosok>
```

A másik fontos művelet az aggregált értékek lekérdezése. Ehhez egy sor aggregációs függvény áll rendelkezésre. A XQuery aggregációs függvénykezelésének sajátossága, hogy itt a függvény argumentumában explicite ki kell jelölni a feldolgozandó halmazt. A definiált fontosabb aggregációs függvények:

- max(): maximum érték,
- min(): minimum érték,
- sum(): összeg,
- count(): darabszám,
- avg():átlag.

A függvények használatára vegyük az alábbi példát: autótípusonként írassuk ki a hozzá tartozó autókat:

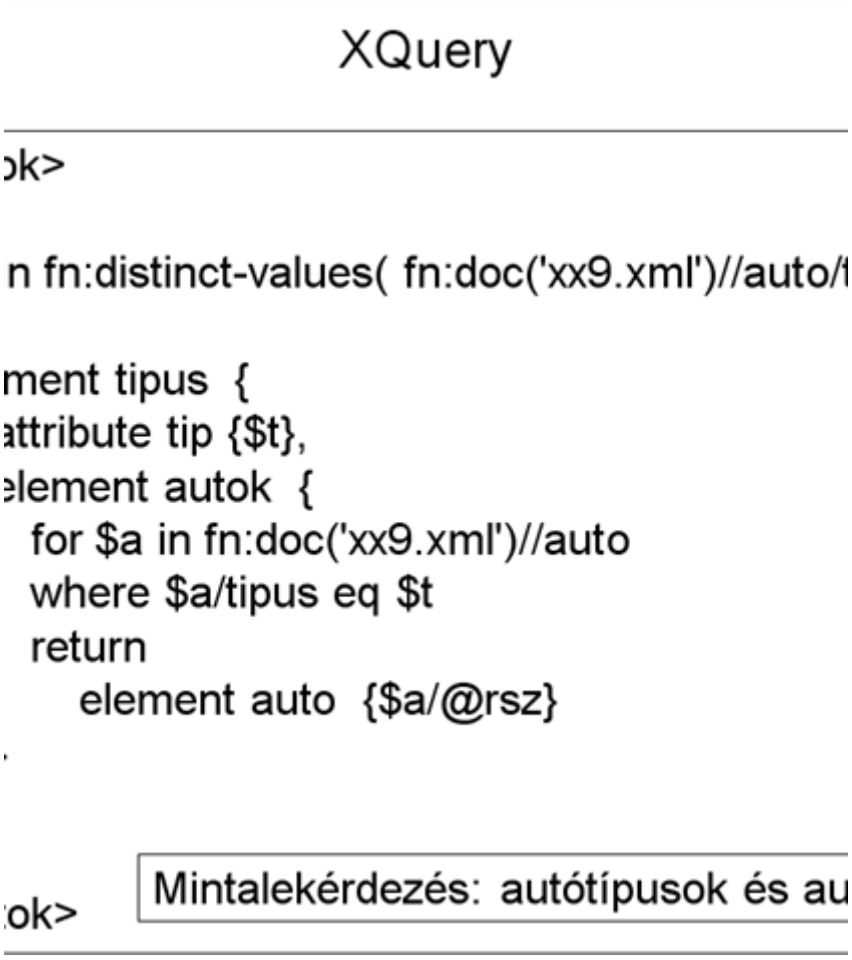
```
  <adatok>
  {
```

```
  for $t in fn:distinct-values( fn:doc('minta.xml')//auto/tipus )
  return
    element tipus {attribute tip {$t},
       element autok {
          for $a in fn:doc('minta.xml')//auto
          where $a/tipus eq $t
          return
            element auto  {$a/@rsz}
       }
    }
}
</adatok>
```

A példában a beágyazott lekérdezés is megfigyelhető, hiszen a külső lekérdezés RETURN részében egy másik FLOWER lekérdezés foglal helyet. A belső lekérdezésben felhasználjuk a külső lekérdezésben létrehozott \$t változót, illetve egy saját független változót is, az \$a-t is definiáljuk. Ha a lekérdezésben az autók tételes felsorolása helyett csak a darabszámukat igényeljük, akkor a lekérdezés alakja a következőre változik:

```
<adatok>
{
for $t in fn:distinct-values( fn:doc('minta.xml')//auto/tipus )
return
   element tipus { attribute tip {$t},
      attribute db {count(
        for $a in fn:doc('minta.xml')//auto
        where $a/tipus eq $t
        return
          element auto  {$a/@rsz}
    )
   }
  }
}
</adatok>
```

Ebben az esetben a darabszámot a count() függvénnyel kérdezzük le, melynek argumentumába megadjuk a megszámlálandó csomóponthalmazt. A csomóponthalmazt most egy újabb FLOWER lekérdezéssel állítjuk elő.



19. Az xQuery minta lekérdezés

2.9. Kvantorok használata

Az XQuery nyelvnek egy saját formátuma van a létezik és minden kvantorokhoz kötött logikai kifejezések leírására. A leírásban meg lehet adni a kötött változót, az alaphalmazt és a feltételt az alábbi alakokban:

```
every $valtozo in halmaz statisfies kifejezés
some $valtozo in halmaz statisfies kifejezés
```

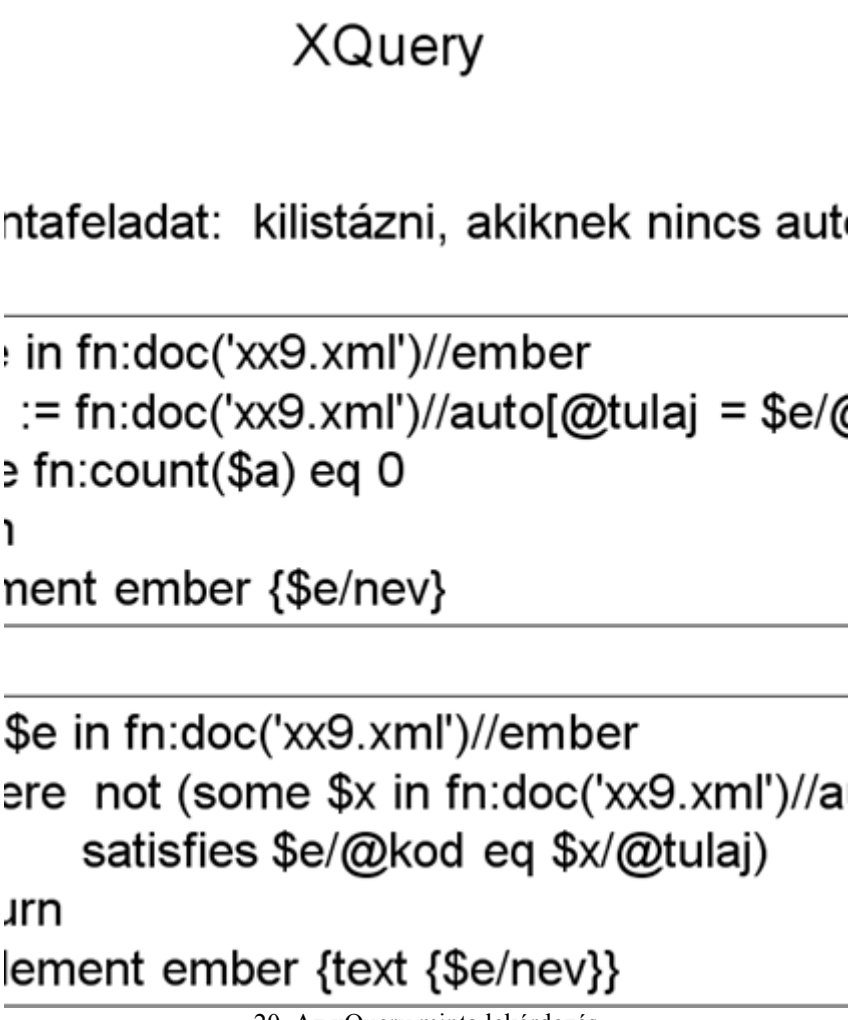
Az every szó a minden kvantorhoz, a some szó a létezik kvantorhoz kötött. A kifejezésben rendszerint szerepel a kötött változó is. Mintaként vegyük azt a kifejezést, amely megadja, hogy létezik-e 200-nál drágább autó:

```
let $oa := fn:doc("minta.xml")/adatbazis/autok
return
  <ered>
    if ({some $a in  $oa/auto statisfies {$a/ar gt 200 }})
    then {element valasz van}
    else {element valasz nincs}
   </ered>
```

A következő példában megkeressük azon embereket, akiknek nincs autója:

```
for $e in fn:doc('minta.xml')//ember
where  not (some $x in fn:doc('minta.xml')//auto
   satisfies $e/@kod eq $x/@tulaj)
 return
   element ember {text {$e/nev}}
```

A fenti lekérdezésben a \$e változó mögé felvettük az ember csomópontok halmazát. Szűrési feltételben ellenőrizzük, hogy az autó csomópontok halmazában, melynek egy elemét a \$x változó hordozza, van-e olyan autó, melynek tulaj elemjellemzője megegyezik a vizsgált ember csomópont kód elemjellemzőjének értékével. Ha nem létezik ilyen autó, akkor az ember neve kikerül az eredmény dokumentumba egy ember nevű csomópont szövegértékeként.



20. Az xQuery minta lekérdezés

A fenti lekérdezést természetesen más módon is meg lehet oldani, az alábbi megoldás az aggregációs függvényekre épít:

```
for $e in fn:doc('minta.xml')//ember
let $a := fn:doc('minta.xml')//auto[@tulaj = $e/@kod]
where fn:count($a) eq 0
return
  element ember {$e/nev}
```

2.10. Adattípus kezelés

A XQuery épít az XMLSchema típusrendszerére, ezért nem lényegtelen az egyes kifejezések típusának ismerete és esetleg az adattípusának az illesztése. Itt is rendelkezésre áll egy típuskonverziós lehetőség, melyet a

```
kifejezés cast as típus
```

parancs biztosít. A kifejezésnek egy elemi kifejezésnek kell lennie, rendszerint egy változót jelöl. Típusként a XQuery és XMLSchema által elfogadott adattípusok jöhetnek szóba. A cast szerkezet egy új értéket generál, melynek adattípusa megegyezik az igényelt adattípussal, tartalma pedig a megadott operandusból származódik. A következő példa az elem ár mezőjét mint egész számot értelmezi:

```
for $x in fn:doc("minta.xml")//auto
return
  {element a {($x/ar cast as xs:integer) * 220} }
```

Az adattípus beállítása mellett lehetőség van egy kifejezés aktuális adattípusának a lekérdezésére is. Ehhez a

```
kifejezés instance of típus
```

formulát kell használni. Ez a formula akkor ad logikai igaz értéket vissza, ha a megadott kifejezés típusa megegyezik a megadott adattípussal, különben a logikai hamis értéket szolgáltatja. Például a

```
<a> 5  </a> instance of xs:integer
```

kifejezés logikai hamis értékű lesz, mivel a baloldali openandus egy csomópont típusú és nem egészszám típusú. Ezzel szemben a

```
<a> 5  </a> instance of element()
```

már logikai igaz értékű lesz.

2.11. Saját függvények definiálása

Az XSLT nyelvhez hasonlóan az XQuery nyelv is támogatja az egyedi függvények létrehozását. Azonban itt is teljesül az a keretfeltétel, hogy a kódot a funkcionális programozás elvei mentén kell készíteni, tehát sem a hagyományos programozási nyelvekben megismert vezérlési szerkezetek sem a változókezelés megszokott mechanizmusai nem állnak rendelkezésre. A változók itt is csak egyszer vehetnek fel értéket és a ciklusok helyett rekurzív függvényhívásokkal lehet megoldani az iterációkat. A függvények létrehozásához itt is szükség van saját névtér kijelölésre. A névtér és függvény létrehozás keretutasításai:

```
declare namespace prefix=kifejezes;
declare function prefix:fnev ($p1 as t1,..) as rtip
```

```
{
  utasitások…
  return kifejezés
};
```

A return utasítás szolgál a függvény visszatérési értéknek megadására. A következő példában egy olyan függvényt készítünk, mely egy decimális, tízes számrendszerbeli számot vesz át az argumentumában és annak a felét adja vissza.

```
declare namespace ll="http:me.kl";

declare function ll:felez($x as xs:decimal) as xs:decimal
{
   let $c:=2
   return $x div $c
};

for $x in doc('minta.xml')/adatbazis/autok/auto
where $x/ar < 222
order by $x/tipus descending
return
   <car> {$x/@rsz} {$x/tipus/text()} {$x/ar} {ll:felez($x/ar)}</car>
```

A mintában a függvény meghívását is láthatjuk. A kifejezés kiértékelés operátorán belül használhatjuk, ahol megadjuk a hívás aktuális paramétereit is. Példánkban az aktuális autó csomópont ár gyerekelemének az értéke kerül át a felez nevű függvényhez. A paraméterátadás során egy automatikus típuskonverziót is végzett a rendszer. Egy iterációt is tartalmazó mintaként vegyük azt a feladatot, amikor az első n szám összegét kell összeadni. Ugyan tudjuk, hogy ez zárt alakban is megadható, de most a gyalogmódszerrel elvégezzük az elemek egyenkénti összeadását:

```
declare namespace ll="http:me.kl";
declare function ll:ossz($r as xs:integer) as xs:integer
{
  let $y := 1
  return  (
     if ($r > 0) then
        $r +  ll:ossz($r -1)
     else
        0
 )
};

let $x := 5
return
   <ered> {ll:ossz($x)}</ered>
```

Az ossz nevű függvényben az \$y azonosítójú változó tulajdonképpen csak helypótló szerepet tölt be, mivel értékadás nélküli return tagot nem fogad el az XQuery szintaktika. Mint itt is látható a ciklus, iteráció egy rekurzív hívási szerkezettel oldható meg. Egy összetettebb feladatként matematikai problémát veszünk, meg kell határozni az $exp(x)$ függvény értékét a hatványsorának segítségével. Mivel itt is iterációra van szükség, s ezt

rekurzióval lehet megoldani, szükség van egy belső rekurzióhoz felhasználható függvényre és egy interface függvényre, melyet a felhasználók hívnak meg. A két függvény kapcsolata az alábbi összefüggésen alapszik:

```
exp(x) = 1 + exp2(x, 1, s, n, m)

exp2(x, i, s, n, m) =
  \frac{s*x}{n*i}  + exp2(x,i+1,s*x, n*i, m),  ha   i<m
```

3. Adatkezelő funkciókkal való kibővítés

Mint azt már e téma bevezető részében is említettük, az XQuery nyelv, mint általános magas szintű XML adatkezelő nyelv egyik leglényegesebb hiányossága, hogy csak az adatok lekérdezését támogatja. Ezen megkötés nagyban gátolja a nyelv univerzális elterjedését. Egy adatbázis kezeléshez, az SQL nyelvhez szokott fejlesztőnek sokkal természetesebbnek tűnik egy olyan általános nyelv, melyben az adatlekérdezés mellett adatkezelő, módosítást végző utasítások is rendelkezésre állnak. Ezen hiányosságok nyilvánvaló volta miatt a XQuery csapat is szükségét érezte az adatkezelés irányába történő továbblépésnek. Az ez irányú fejlesztésekre megindult a kidolgozó munka, s ugyan még nem jelent meg hivatalos szabvány az adatmódosításra, de egy igen alaposan kidolgozott munkapéldány már napvilágot látott 2007 őszén. Az elkészített szabványvázlat az XQuery Update Facility nevet viseli. A következőkben röviden összefoglaljuk a 2007-es munkaváltozatban megfogalmazott irányelveket, s a parancsok szintaktikájára adott javaslatokat. A módosítások koncepcionális modellje, mint ahogy a lekérdezési műveletek modellje, az XML dokumentum XDM reprezentációján alapszik.

Az XQuery Update nyelv feldolgozási modelljét illető legfontosabb változás, hogy a végrehajtó motorban a kifejezések eredménye nemcsak XDM (vagy Infoset) kimeneti atom lehet, hanem egy módosítást leíró információlista lista is (Update List). A módosítási lista célja megadni, hogy milyen módosításokat kell majd elvégezni az egyes csomópontoknál. A módosítási lista elemei az úgynevezett módosítási atomok (Update Primitives), melyek mindegyike egy önállóan elvégezhető módosítást ír le. A módosítási atom két komponense:

- Célcsomópont, amin el kell végezni a módosítást,
- Művelet, az elvégzendő tevékenységatom.

A tevékenységatom egy belsőleg értelmezett átalakítást ad meg, a felhasználó adatmódosítást előíró műveletei ezen tevékenységatomokra épülnek. A XQuery modell az alábbi tevékenységatomokat definiálja:

- insertBefore: csomópont beszúrása egy másik csomópont elé,
- insertAfter: csomópont beszúrása egy másik csomópont mögé,
- insertInto: csomópont beszúrása egy másik gyerekei közé,
- insertIntoAsFirst: csomópont beszúrása egy másik első gyerekeként,
- insertIntoAsLast: csomópont beszúrása egy másik utolsó gyerekeként,
- insertAttributes: elemjellemző beszúrása,
- delete: csomópont törlése,
- replaceNode: csomópont helyettesítése egy másik csomóponttal,
- replaceValue: csomópont szövegértékének helyettesítése,
- replaceElementContent: csomópont tartalmának helyettesítése,
- rename: csomópont nevének megváltoztatása.

A fenti atomokból összetett felhasználó parancsokat képeztek, melyek már kiadhatók az XQuery környezetben. A módosításhoz kapcsolódó műveleteknek öt típusa van:

- insert (beszúrás, bővítés)
- delete (törlés),
- replace (helyettesítés),
- rename (átnevezés),
- transform (átalakítás).

A fenti parancsok csomóponthalmazokat illetve egyedi csomópontot kapnak operandusként, s hatásukra a célcsomópontok szerkezete, sémája megváltozhat. A végrehajtó motor a parancsokat felbontja tevékenységatomokra egy módosítási listát eredményezve. A végrehajtás záró fázisában a motor sorban végrehajtja a módosítási listában megadott műveleteket. A következőkben sorba vesszük az egyes parancsok szintaktikája. A parancsokhoz tartozó példákban is a korábban használt minta.xml forrásra építkezünk.

A csomópontfelvitelnek három fontosabb szintakszis változata van. A gyerekcsomópont felvitelének alakja:

```
insert nodes forras_csomopont as (first|last) into  cel_csomopont
```

Példaként vigyünk fel egy új, autót leíró elemet:

```
insert nodes <auto rsz="r5">  <tipus> Opel</tipus> <ar>365 </ar> </auto>
  as last into  fn:doc("minta.xml")/adatbazis/autok
```

Az elem elé történő beszúrás parancsának alakja:

```
insert node forras_csomopont  before cel_csomopont
```

Nézzük meg, hogyan lehet minden autó leíró csomópontot kiegészíteni egy évjárat elemmel:

```
for $v in fn:doc("minta.xml")//ar
return
   insert nodes <evjarat> 1999</evjarat> before $v
```

Egy új csomópontnak egy magadott másik elem mögé történő beszúrásának utasítása az alábbi alakban adható meg:

```
insert node forras_csomopont after  cel_csomopont
```

Elem csomópont törlése a delete paranccsal lehetséges:

```
delete node  cel_csomopont
```

Példaként töröljük ki az Opel autók árát leíró elemet:

```
delete node fn:doc("minta.xml")//auto[tipus = "Opel"]/ar
```

Csomópontok cseréjének utasítása:

```
replace node cel_csomopont with  forras_csomopont
```

Elem szöveges tartalmának cseréje:

```
replace value of node cel_csomopont with  forras_csomopont
```

A példában megnöveljük a Skoda autók ártá 10%-kal:

```
replace value of node fn:doc("minta.xml")//auto[tipus="Skoda"]/ar
with fn:doc("minta.xml")//auto[tipus="Skoda"]/ar*1.1
```

Elem nevének módosítása:

```
 rename node cel_csomopont as nev
```

Elemek átmásolása tartalom módosítással:

```
copy $v := cel_csomopont modify modosito_kifejezes return eredmeny
```

Az alábbi példában kiíratjuk az autók adatait az ár elem nélkül:

```
for $a in fn:doc("minta.xml")//auto
return
  copy $v := $a
  modify delete node $v/ar
  return $v
```

Az utolsóként megadott szintaxis mutatja, hogy az XQuery Update nyelvben különbséget kell tenni módosító és nem-módosító (lekérdező) kifejezések között, hiszen bizonyos esetekben, mint például a FLOWER kifejezés FOR, LET, WHERE és ORDER tagjában csak nem módosító kifejezés szerepelhet. Adatmódosító parancs csak a RETURN tagban adható meg.

Az XQuery egyedi formátumú lekérdező felülete mellett kidolgozták az XQuery XML formátumú alakját is, az XQueryX nyelvet. Ez a formátum lényegesen bőbeszédűbb, hosszabb az eredeti formátummal összevetve. A szabvány leírásban található mintaprogram például 25-ször több sorból áll XQueryX parancs mint a normál XQuery parancs. Ennek a bőbeszédűségnek az az oka, hogy az XQueryX tulajdonképpen magát az XQuery parancsot írja le XML formátumban kifejtve. Például a

```
for $a in fn:doc("minta.xml")//auto
```

parancs változójának (\$a) a megadása is terjedelmes leírást eredményez, hiszen részletesen meg kell adni az alkalmazási kontextus minden részletét:

```
<xqx:queryBody>
  <xqx:flowrExp>
     <xqx:forClause>
  <xqx:forClauseItem>
    <xqx:typedVariableBinding>
<xqx:varname>
a
</xqx:varname>
    </xqx:typedVariableBinding>

..
.
```

Az XQuery és az XQuery Update nyelvek tehát egy teljes értékű XML adatkezelő nyelv irányába indultak el, sorsuk és sikerességük azonban várhatólag a többi konkurens nyelvek, mint az XSLT és az XSQL nyelvek elterjedésétől is jelentősen függ.

4. Feladatok, mintapéldák

1. Feladat: Állítsa elő az autók rendszámait visszaadó listát!

   Megoldás:

```
for $r in fn:doc("gyak1.xml")//auto/@rsz
return
 <rsz>{$r}</rsz>
```

2. Feladat: Állítsa elő az autók rendszámát és árát az ár szerinti sorrendben előállító listát!

   Megoldás:

```
for $r in fn:doc("gyak1.xml")//auto
order by $r/ar
return
  <a>{text {$r/@rsz} }: {$r/ar/text()}</a>
```

3. Feladat: Kérdezze le mennyi autó drágább mint 30000!

   Megoldás:

```
for $r in fn:doc("gyak1.xml")
return
  count($r//auto[number(ar) > 30000])
```

4. Feladat: Állítsa elő a miskolci tulajdonosok autóinak rendszámait tartalmazó listát!

   Megoldás:

```
for $r in fn:doc("gyak1.xml")//auto
 where $r/tulaj/varos = 'Miskolc'
 return
   text{$r/@rsz}
```

5. Feladat: Adja meg az autótípusokat és példányaik darabszámát példányszám szerint csökkenő sorrendben!

   Megoldás:

```
let $d:=fn:doc("gyak1.xml")
 for $r in  fn:distinct-values(fn:doc("gyak1.xml")//auto/tipus/text())
 order by count($d//auto[ tipus/text() = $r]) descending
 return
   <tipus>
     <tip>{$r}</tip><db>{count($d//auto[ tipus/text() = $r])} </db>
   </tipus>
```

6. Feladat: Adja meg városonként mennyi az ottani autók darabszáma és összára csökkenő sorrendben!

   Megoldás:

```
let $d:=fn:doc("gyak1.xml")
for $r in  fn:distinct-values(fn:doc("gyak1.xml")//varos/text())
return
  <varos>
     <nev>{$r}</nev>
      <db>{count($d//auto[tulaj/varos/text() = $r])}</db>
      <osszar>{sum($d//auto[ tulaj/varos/text() = $r]/ar)} </osszar>
  </varos>
```

7. Feladat. A Peter nevű emberhez egy új Skoda autót felvitele.

   Megoldás:

```
for $a in fn:doc("minta.xml")//ember[nev="Peter"]/@kod
return
   insert node
     <auto rsz+"r23">
        {attribute tulaj {$a}}
        <tipus>Skoda</tipus>
      </auto>
   into fn:doc("minta.doc")/adatbazis/autok
```

8. Feladat. Peter autóinak árát az eddigi maximum ár + 1 értékre módosítjuk.

   Megoldás

```
let $m := max(fn:doc("minta.xml")//auto/ar)
return
replace value of node fn:doc("minta.xml")//ember[nev="Peter"]/ar
  with {$m+1}
```

9. Feladat. Az átlagárnál olcsóbb autók árának megnövelése 10%-kal.

   Megoldás:

```
let $autok := max(fn:doc("minta.xml")/adatbazis/autok
let $r := avg($autok/auto/ar)
for $a in $autok/auto/ar
where $a le $r
return
   replace value of node $a with {data($a)+1}
```

10. Feladat. Azon emberek kitörlése, akiknek nincs autójuk:

    Megoldás:

```
let $aa := fn:doc("minta.xml")/adatbazis
for $e in $aa//ember
let $c := count($aa//auto[@tulaj = $e/kod])
where $c eq 0
return
  delete node $e
```

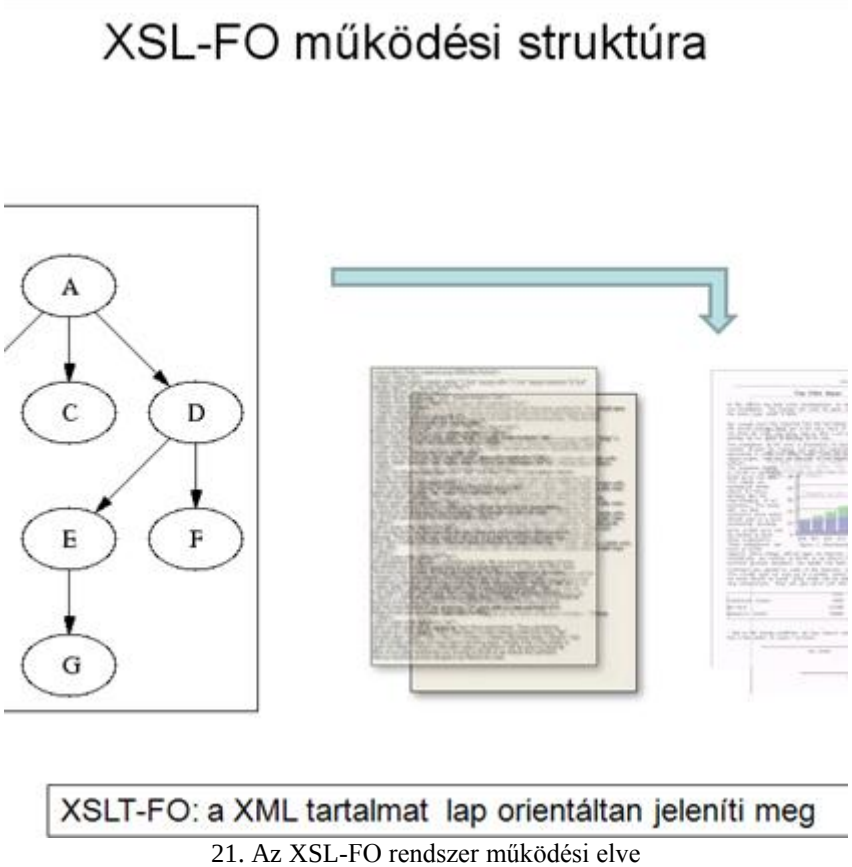
A fejezet végén egy rövid demo szolgál a működési elvek alapelveinek átismétlésére: **Xquery működési mechanizmus demo animáció**.

3. fejezet - Az XSL-FO nyelv elemei

1. Az XSL-FO nyelv elemei

Az XML formátum a jellegéből fakadóan általános leíró nyelvként használható, melynek egyik fő előnye, hogy több feldolgozó szabvány nyelv is kapcsolódik hozzá. Az univerzalitás jegyében első esetünket az XSL-FO nyelv bemutatására szenteljük. Az XSL-FO nyelv a nyomtatott dokumentumok definiálására, leírására szolgáló nyelv. Az FO tag a Formatting Object, azaz formátum leíró nyelv témakijelölésre utal. A XSL-FO a HTML nyelvtől eltérően a lapokra tagolt dokumentumok specifikációs nyelve, ezért itt több, a lap formai elemeihez kapcsolódó elemet hoz be a szabvány. Az XSL-FO nyelv illeszkedik az XML nyelv szabályaira, tehát ő maga is viszonylag könnyen konvertálható, előállítható más XML dokumentumokból. Az XSL-FO feldolgozó motor a XSL-FO formátumú forrás állományból valamilyen nyomtatható formátumú állományt generál. A támogatott típusok között szokott lenni a PDF, PS és RTF formátum.

Az XSL-FO fő célja a megjelenítési formátumot lehetőség szerint eszköz független módon specifikálni a lapokra osztható kimenetek esetére. A rendszer bemeneti dokumentumának XML formátumban kell lennie. Olyan XML dokumentumnak felel meg az XSL-FO forrás, melyben speciális XML elemek, névterek és elemjellemzők állnak rendelkezésre a megjelenítési szerkezet definiálására.



21. Az XSL-FO rendszer működési elve

Az XSL-FO nyelvben a dokumentum gyökéreleme egy root elem, melynek formátuma és névtere a következő:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <!-- The full XSL-FO document goes here -->
</fo:root>
```

A dokumentum modelljében központi helyet foglal el a lap fogalma, illetve a lapon megjelenő további formátummal rendelkező elem. Ezen elemekből a fontosabbakat foglaljuk össze a következő táblázatban.

lap (page)
: A nyomtatás egysége. A lap megtervezésekor ki lehet térni a lap paramétereire, mint a méret, irányítás, margók, fejléc és lábléc. A lapok szerkezetének megadásakor lehetőség van sémák definiálásáa.

inline-elem
: A soron belül megjelenő, egyedi külalakkal rendelkező rész.

blokk (block) elem
: A lapon megjelenő, négyzet területtel lefedhető objektumokat fogja össze. Ide tartozik többek között a lista, a kép vagy a bekezdés.

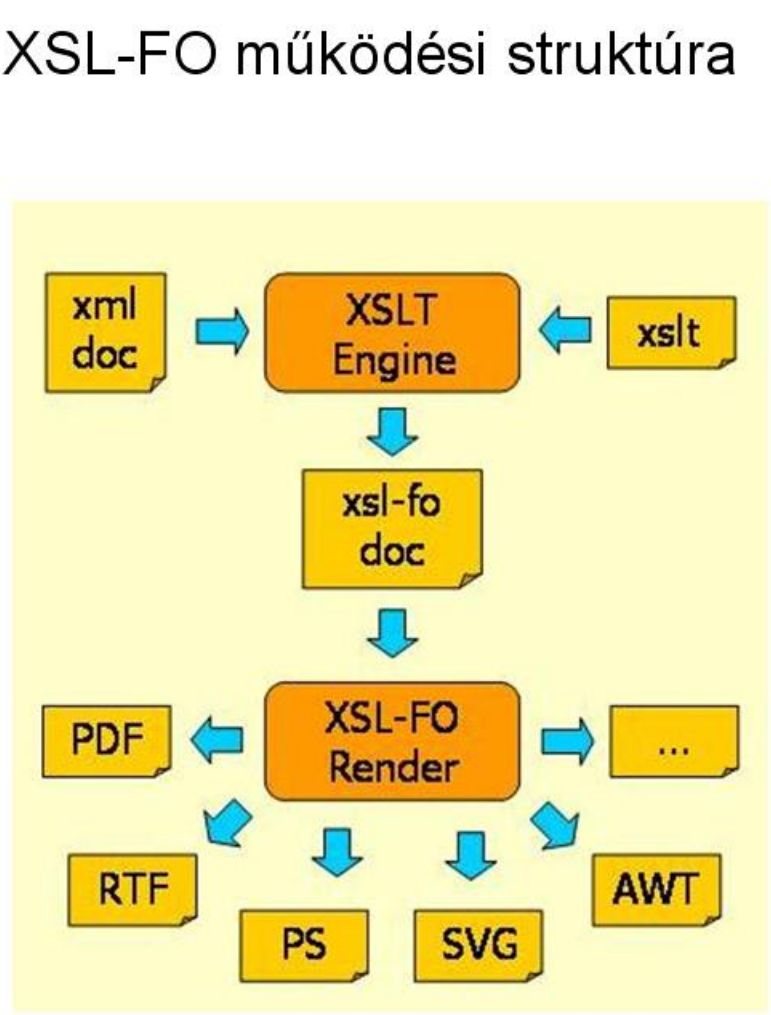
terület (area)
: A lap egy szegmense.

flow
: A lapszegmensek tartalommal való feltöltésekor megjelenő sorrend.

lap leíró séma (page-master)
: Egy lap teljes megjelenítési formátumát nyilvántartó egység.

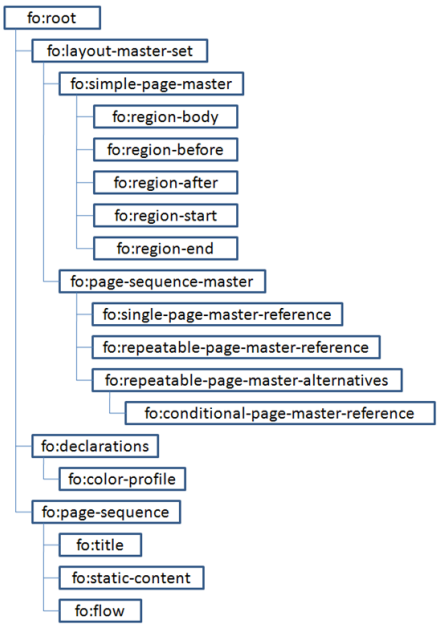
lapok sorrendisége (page sequence)
: A lapok egymásutániságát szabályozó rész.

régió (region)
: A lap egy saját megjelenítési paraméterkészlettel rendelkező része.



22. Az XSL-FO feldolgozás architektúrája

Az XSL-FO dokumentumok egy igen pontosan rögzített szerkezettel rendelkeznek. A leíró rész első felében a lapok alapparamétereit lehet beállítani, majd a lap egyes fő blokkjainak a megadása történik. Ezt követően a lapok egymásutániságához, a lapsorozathoz kötődő paraméterek következnek. A harmadik szegmens a lapok tartalommal történő feltöltéséhez kapcsolódik. A dokumentum elemhierarchiáját szemlélteti az alábbi ábra.



23. Az XSL-FO dokumentum elem hierarchiája

Az XSL-FO leírás alapvetően két fő részből áll. Az első részben a lapok sémáját lehet definiálni. A generáláskor több lapséma is felhasználásra kerülhet a dokumentum különböző részeiben, A lapsémákat a

```
<fo:layout-master-set>
      ....
</fo:layout-master-set>
.
```

keret határolja. A leírás második részében a lapok sorrendiségét lehet megadni. Itt lehet a forrás adattartalmához lapsémákat hozzárendelni. Ezen részt a

```
<fo:page-sequence>
      ....
</fo:page-sequence>
.
```

szerkezet fogja közre. A lap fő szerkezetét a

```
<fo:simple-page-master>
      ....
</fo:simple-page-master>
.
```

elem tartalmazza. Egy ilyen leíró egységhez egy azonosító név is társul, tehát az XSL-FO dokumentumban több lap definíciós séma adható meg. A lapséma fő paraméterei:

- lap magassága,
- lap szélessége,
- felső margó,
- alsó margó,
- jobb oldali margó,
- baloldali margó,
- keret formátuma.

A formai elemek meghatározásakor gyakran adunk meg méret értékeket, mint például a margó szélessége. A méretek esetében a számérték mellett fontos információ a mértékegység is. A XSL-FO rendszerben is többféle mértékegység is kijelölhető. Az alábbi lista összefoglalja a támogatott mértékegységek körét.

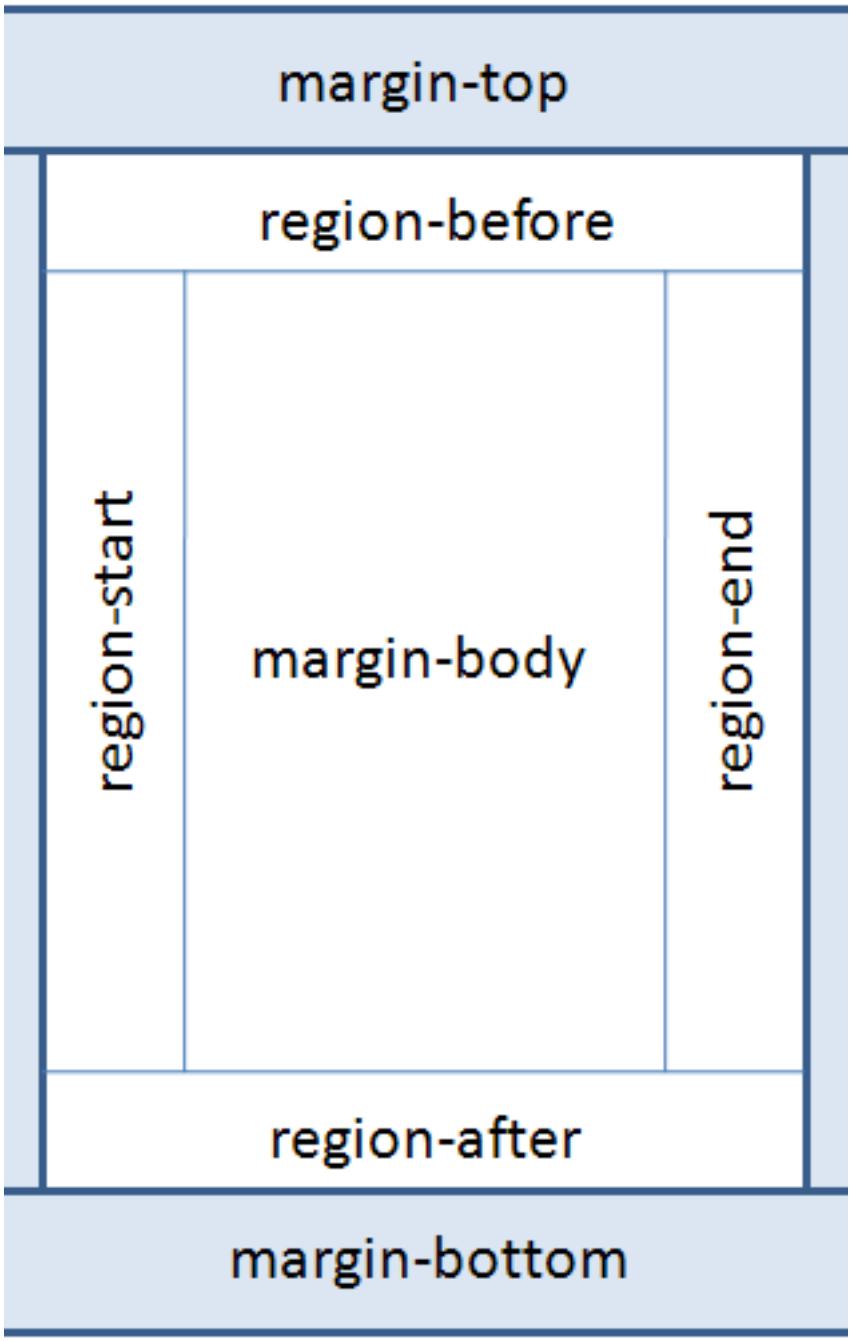
- mm : milliméter,
- cm: centiméter,
- in: inch, hüvely,
- pt: pont (1/72 inch),

- px: pixel,
- %: százalékos arány.

A margók által határolt terület az írható rész. Az írható rész tovább bontható kisebb, önállóan paraméterezhető részekre, úgynevezett tartományokra, régiókra. A fontosabb régiók közé tartozik a

- region-body: a lap törzsrésze,
- region-before: a lap fejrésze,
- region-after: a lap lábjegyzete,
- region-start: baloldali margó,
- region-end. a jobboldali margórész.

Az egyes laprészek elhelyezkedését szemlélteti a következő ábra.



24. Az XSL-FO lap szerkezete

Az egyes régiók át is lapolhatják egymást, tehát a tervező feladat gondoskodni a megfelelő térbeli elrendezésről. Egy-egy régiót szövegtartalommal lehet feltölteni. A régió egyik jellemzője, hogy hogyan kezeli a túl hosszú szövegeket. Több lehetséges alternatíva lézetik, mint csonkítás, görgetés.

A régió kezelés egyik sajátossága, hogy a region-body és többi régió alapesetben átfedi egymást, azaz ugyanazt a laprészt foglalják le. Ez sokszor zavaró tulajdonság, de néha hasznos is lehet. Így lehet például két szöveget egymásra vetíteni. Ez történik például akkor, amikor minden lapra fixen ráviszünk egy copy-right üzenetet. A területek átfedésének megszüntetéséhez be kell állítani az egyes régiók méretét és margó határait. A két fő lépés az alábbi attribútumokkal oldható meg, ahol most csak a felső szegély beállítására szorítkozunk:

- A törzs terület szegélyének meghatározása, ez adja meg az írható terület felső határát: <fo:region-body margin-bottom="1cm"

- A region-before terület szegélyének meghatározása, ez adja meg az region-before rész írható területének alsó határát: <fo:region-before extent="1cm"

Akkor lehet azt átfedést biztosan kikerülni, ha az extent-ben megadott érték kisebb, mint a margin attribútumnál szereplő érték.

A lapok tartalmát meghatározó page-sequence rész tartalmazhat:

- stattkus tartalmat: olyan rész, amely minden lapon ugyanúgy jelenik meg, és
- dinamikus (flow) elemeket, mely a változó tartalmat jeleníti meg.

A dinamikus tartalomfeltöltés a flow elemmel definiálható. Ezen elem egyik elemjellemzője megadja, hogy mely lapsémát kell felhasználni az eredmény előállításakor. Ezen parancsot szemlélteti az alábbi példa:

```
<fo:page-sequence master="lapséma">
  <fo:flow flow-name="nev">
  .. tartalom
 </fo:flow>
</fo:page-sequence>
.
```

A dokumentum tartalma blokkokba foglalva kerül át a lapokra. A blokkot befoglaló elem:

```
<fo:block>
      ....
</fo:block> .
```

A blokk egy általános konténer, mely tartalmazhat többek között paragrafust, sorokat, címsort, feliratot, megjegyzéseket. A blokkhoz kapcsolódóan be lehet állítani a megjelenítési paramétereket, mint

- Font jellemzői (típus, méret, ..),
- Szín beállítások,
- Szóközök, szótávok beállítása,
- Margók, beugrások.

A blokk beállítására ad példát az alábbi minta:

```
<fo:block
   text-align="start"
   margin-left="1.1in"
   space-after="6pt"
   line-height="13pt"
   font-family="sans-serif"
   font-weight="bold"
   font-size="12pt"
   font-height="10pt"
   font-weight="bold">
       Itt az üzenet
</fo:block>
```

A blokkok egyik alapparamétere a határoló keret beállítása. Általánosságban elmondható, hogy a beállítás az

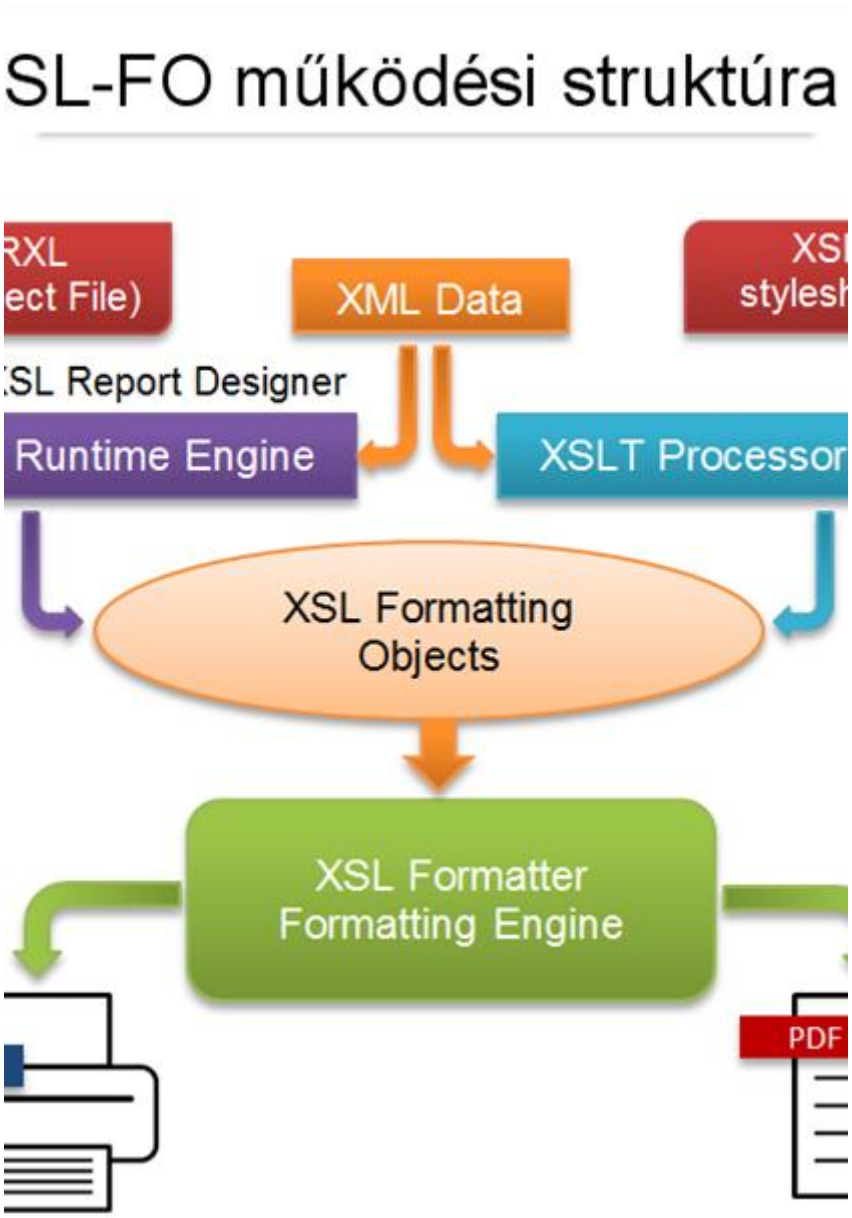
```
border = "szélesség"  "stílus"  "szín"
```

alakban lehetséges

A lapok kiosztását rugalmasan lehet kezelni, többek között lehetőség van arra, hogy eltérő lapsémát rendeljük az első laphoz, illetve bármelyik tetszőleges laphoz. Ezen hozzárendelést a

```
<fo:page-sequence-master>
      ....
</fo:page-sequence-master>
```

.



25. Az XSL-FO feldolgozás architektúrája

A következő példában egy teljesebb lap definíciót mutatunk be

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="LetterPage" page-width="5in"
      page-height="2in">
      <fo:region-body region-name="PageBody" margin="0.2in"/>
   </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="LetterPage">
    <fo:flow flow-name="PageBody" font-family="Arial" font-size="10pt">(1)
      <fo:block text-align="justify" space-after="0.1cm" (2)
       border="3pt solid green" padding="5pt">
        This is the first paragraph of justified text.
         Notice how text fills all available space for all lines
         except the last one.The alignment of the last line is
         controlled by text-align-last property.
      </fo:block>
      <fo:block space-before="1cm" border="3pt ridge blue" padding="5pt">
        This is the second paragraph. This block is left aligned.
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Az XSL-FO rendszerben egy dokumentumhoz több lapséma is kapcsolódhat. Erre szükség is van, hiszen egy komolyabb dokumentumban több, egymástól eltérő részek foglalnak helyet. A fő lap rendszerint eltér a többi laptól és dokumentum különböző részein az eltérő tartalom eltérő formátumot fog igényelni. Az eltérő lapsémák megadásához a <layout-master-set> elemen belül több <simple-page-master> leíró elemet adunk meg. Az egyes lapleíró sémákat egyedi névvel azonosíthatjuk, mely azonosító nevet a master-name elnevezésű attribútumban adunk meg. A sémarészben ekkor egy összefogó sémát is megadhatunk, melyet a <page-sequence-master> elem alatt kell szerepeltetni. Ezen részben lehet megadni egy lapra illetve több lapra érvényes séma hozzá rendelést, azaz definiálhatjuk a lapsémák lapokhoz történő rendelését. Az egy lapra érvényes összerendeléshez a <single-page-master-reference> elemet, míg a több lapra érvényes összerendelést a <repeatable-page-master-reference> elemmel írhatjuk le. A tartalmat leíró részben továbbra is a master-reference attribútummal jelölhetjük ki a kapcsolódó séma leírást. A következőkben egy példát mutatunk be a többsémás dokumentum megvalósítására:

```
<fo:root>
 <fo:layout-master-set>
   <fo:simple-page-master  master-name="m1">
      <fo:region-body margin="2cm">...
    </fo:simple-page-master>
    <fo:simple-page-master  master-name="m2">
      <fo:region-body margin="4cm" border="thin">...
    </fo:simple-page-master>
    <fo:page-sequence-master master-name="mm">
      <fo:single-page-master-reference   master-reference="m1" />
      <fo:repeatable-page-master-reference   master-reference="m2" />
 </fo:layout-master-set>

 <fo:page-sequence master-reference="mm">
    <fo:flow flow-name="xsl-region-body">
      <fo:block break-after="page" font="..."  >
        Folap ...
```

```
        </fo:block>
        <fo:block  font="..."  >
           Második lap ...
        </fo:block>
     </fo:flow>
  </fo:page-sequence>
</fo:layout-master-set>

</fo:root>
```

A tartalmat meghatározó részben szereplő - <fo:flow> elemhez tartozik egy flow-name attribútum is. Ez a név egyik fontos feladata, hogy kijelöli azt a tartományt, ahová a szöveget le kell tenni. A következő előre definiált nevek használhatók:

- xls-region-body : region-body régió,
- xls-region-before : region-before régió,
- xls-region-after : region-after régió,
- xls-region-start : region-start régió,
- xls-region-end : region-end régió.

A lapsorozat használatához kapcsolódóan néhány további elemet emelünk ki az alábbi listában:

- A <fo:repeatable-page-master-reference> elemnél megadható egy maximum-repeats attribútum, mely meghatározza, maximum mennyi ismétlése lehet az aktuális lapsémának.
- Az összetettebb dokumentumoknál nem lehet előre tudni az egyes formátumhoz tartozó lapok lapsorszámát. Emiatt igény jelentkezik egy rugalmasabb, dinamikusabb lapséma kiosztásra is. A rugalmasabb feltételhez kötött lapséma kijelölést a

  ```
  <fo:repeatable-page-master-alternatives>
    feltételes séma kijelölés
   </fo:repeatable-page-master-alternatives>
  ```

  paranccsal lehet meghatározni. A feltételes kijelölés a

  ```
  <fo:conditional-page-master-reference  master-reference="cim" feltétel"/>
  ```

  gyerekelemmel lehetséges.

- A feltételes lapséma kijelölésnél az alábbi feltétel elemek adhatók meg:
  - page-position: lap sorszámára vonatkozó feltételt vizsgál,
  - odd-or-even: a lapsorszám páros vagy páratlan voltát vizsgálja,
  - blank-or-not-blank: a lap tartalmának üres vagy nem üres voltát ellenőrzi.

- A lapok sorszámát a generátor program automatikusan állítja elő, viszont lehetőség van bizonyos beavatkozásra ebben a sorszámgenerálásban. Kétféle módosításra van lehetőség, mindkettőt a <fo:page-sequence> elemnél, annak attribútumaként lehet megadni.
  - initial-page-number : az induló lapsorszám,
  - force-page-cont : megadható, hogy a dokumentum rész, kötelezően páros vagy páratlan oldalszámra végződjön.
- A lapok tartalmának több oszlopba történő bontását a <fo:region-body> elemnél lehet jelezni, ahol egy column-count attribútumot lehet szerepeltetni az oszlop darabszám megadására.
- A tartományon túlfolyó szöveg kezelésére szintén a <fo:region-body> elemnél van lehetőség. Itt megadható egy overflow elnevezésű attribútum, mely az alábbi értékeket veheti fel.
  - visible: látszik a tartalom egy másik blokkba átnyúlva,
  - hidden: elrejti a túlnyúló tartalmat,
  - scroll: ha lehet, görgetési funkciót hoz létre.
- Az egyes bekezdéseket a folyó szövegben a <fo:block> elemmel határoljuk. A block elemnél attribútumként szerepelhetnek olyan formátum leíró elemek, mint
  - space-before: sortáv a bekezdés előtt,
  - space-after: sortáv a bekezdés után,
  - start-indent: bekezdéstáv mértéke a blokk kezdetekor,
  - end-indent: bekezdéstáv mértéke a blokk zárásakor,
  - border: a bekezdés szegélyének megadása.
- A listák megadására szolgálnak az alábbi elemek:
  - <fo:list-block>: a lista keretét megadó elem, ez alatt szerepelnek a listát leíró további elemek,
  - <fo:list-item>: egy listaelemet megadó elem,
  - <fo:list-item-label>: a lista egy eleme előtt álló jelölő elem,például sorszám,
  - <fo:list-item-body>: a lista egy elemének tartalma.
- A táblázatok megadására szolgálnak az alábbi elemek:
  - <fo:table>: a táblázat keretét megadó elem, ez alatt szerepelnek a táblát leíró további elemek,
  - <fo:table-header>: táblázat fejrésze,
  - <fo:table-body>: táblázat adatrésze,
  - <fo:table-row>: a táblázat egy sora,
  - <fo:ltable-cell>: a táblázat egy mezője.
- Képek beültetése a szövegbe a <fo:external-graphic > elemmel lehetséges. Az elem attribútumai közé tartozik forrásfile-t kijelölő src attribútum illetve a méretet meghatározó width és height paraméterek.
- A szövegbe lapsorszámot egy <fo:page-number/> hivatkozással lehet felvinni.

Az XSL-FO természetesen jóval több elemet tartalmaz, mint amennyi elemet mi most át tudtunk tekinteni. A XSL-FO rendszer alapvető szerepet játszik az XML alapú dokumentum készítő rendszerekben, hiszen az XSL-FO motorok konvertálják az XML leírást PDF vagy RTF formátumú kimenetre. Az XSL-FO-ra épülő egyéb

XML dokumentum kezelők rendszerint témakör specifikus elemekkel egészítik ki az alap XSL-FO nyelvet. Az elterjedtebb XML alapú dokumentum kezelők közül megemlíthető a Docbook és a DITA szabvány, melyek kezelése csak XSL-FO és XSLT ismerete mellett lesz igazán hatékony.

2. Feladatok, példák

1. Feladat: Írjon ki egy elemi üzenetet a lapra!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="m1">
      <fo:region-body></fo:region-body>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="m1">
    <fo:flow flow-name="xsl-region-body">
      <fo:block >Hello világ!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

2. Feladat: Készítsen táblázatos adatokat megjelenítő lapot!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xml:lang="en">

  <fo:layout-master-set>
    <fo:simple-page-master page-height="290mm" page-width="210mm"
    margin-top="10mm" margin-left="20mm" margin-right="20mm"
    margin-bottom="10mm" master-name="folap">
      <fo:region-body column-count="1" border-style="none" border-width="thin"
      margin-top="20mm" margin-left="0mm" margin-right="0mm" margin-bottom="20mm"/>
      <fo:region-before border-style="none" border-width="thin" extent="15mm"/>
      <fo:region-after border-style="none" border-width="thin" extent="15mm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence initial-page-number="1" master-reference="folap">

    <fo:static-content flow-name="xsl-region-before">
      <fo:block font-size="9pt">Minta dokumentum</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body">
     <fo:block  text-align="left"  >Táblázat bevezetés</fo:block>
```

```
<fo:table table-layout="fixed" border-width="1mm" border-style="solid">
        <fo:table-column column-width="3in"/>
        <fo:table-column column-width="3in"/>
        <fo:table-header text-align="center" background-color="silver">
          <fo:table-row>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block font-weight="bold">Cim1</fo:block>
            </fo:table-cell>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block font-weight="bold">Cim2</fo:block>
            </fo:table-cell>
          </fo:table-row>
        </fo:table-header>
        <fo:table-body>
          <fo:table-row>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block>cella 1 1</fo:block>
            </fo:table-cell>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block>cella 1 2</fo:block>
            </fo:table-cell>
          </fo:table-row>
          <fo:table-row>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block>cella 2 1</fo:block>
            </fo:table-cell>
            <fo:table-cell padding="1mm" border-width="1mm" border-style="solid">
              <fo:block>cella 2 2</fo:block>
            </fo:table-cell>
          </fo:table-row>
        </fo:table-body>
        </fo:table>

    </fo:flow>
  </fo:page-sequence>
 </fo:root>
```

3. Feladat: Készítsen egy listát tartalmazó dokumentumot!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xml:lang="en">

  <fo:layout-master-set>
    <fo:simple-page-master page-height="290mm" page-width="210mm"
     margin-top="10mm" margin-left="20mm" margin-right="20mm"
     margin-bottom="10mm" master-name="folap">
      <fo:region-body column-count="1" border-style="none" border-width="thin"
      margin-top="20mm" margin-left="0mm" margin-right="0mm" margin-bottom="20mm"/>
      <fo:region-before border-style="none" border-width="thin" extent="15mm"/>
      <fo:region-after border-style="none" border-width="thin" extent="15mm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence initial-page-number="1" master-reference="folap">
```

```
<fo:static-content flow-name="xsl-region-before">
   <fo:block font-size="9pt">Minta dokumentum</fo:block>
</fo:static-content>

<fo:flow flow-name="xsl-region-body">
 <fo:block  text-align="center">Lista bevezetés</fo:block>
   <fo:list-block
      start-indent="5mm"
      provisional-distance-between-starts="10mm">
      <fo:list-item>
         <fo:list-item-label end-indent="label-end( )">
            <fo:block>&#x2022;</fo:block>
         </fo:list-item-label>
         <fo:list-item-body start-indent="body-start( )">
            <fo:block>Lista feje</fo:block>

               <fo:list-block>
                <fo:list-item>
                   <fo:list-item-label
                      end-indent="label-end( )">
                      <fo:block font-family="ZapfDingbats">&#x2798;
                      </fo:block>
                   </fo:list-item-label>
                   <fo:list-item-body
                      start-indent="body-start( )">
                      <fo:block>lista tartalma első elem.
                      </fo:block>
                   </fo:list-item-body>
                </fo:list-item>

                  <fo:list-item>
                     <fo:list-item-label
                        end-indent="label-end( )">
                        <fo:block font-family="ZapfDingbats">&#x2798;
                        </fo:block>
                     </fo:list-item-label>
                     <fo:list-item-body
                        start-indent="body-start( )">
                        <fo:block>lista tartalma mádodik elem.
                        </fo:block>
                     </fo:list-item-body>
                  </fo:list-item>

               </fo:list-block>

         </fo:list-item-body>
      </fo:list-item>
   </fo:list-block>
</fo:flow>
</fo:page-sequence>
</fo:root>
```

4. Feladat: Készítsen egy egy- és kéthasábos lapokat tartalmazó dokumentumot!

   Megoldás:

```
<?xml version="1.0" encoding="UTF-8" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xml:lang="en">

  <fo:layout-master-set>
    <fo:simple-page-master page-height="290mm" page-width="210mm"
    margin-top="10mm" margin-left="20mm" margin-right="20mm"
    margin-bottom="10mm" master-name="folap">
      <fo:region-body column-count="1" border-style="none" border-width="thin"
      margin-top="20mm" margin-left="0mm" margin-right="0mm" margin-bottom="20mm"/>
      <fo:region-before border-style="none" border-width="thin" extent="15mm"/>
      <fo:region-after border-style="none" border-width="thin" extent="15mm"/>
    </fo:simple-page-master>
    <fo:simple-page-master page-height="290mm" page-width="210mm"
    margin-top="10mm" margin-left="20mm" margin-right="20mm"
    margin-bottom="10mm" master-name="lap2">
      <fo:region-body column-count="2" column-gap="20mm" border-style="none"
      border-width="thin" margin-top="20mm" margin-left="0mm" margin-right="0mm"
      margin-bottom="20mm"/>
      <fo:region-before border-style="none" border-width="thin" extent="15mm"/>
      <fo:region-after border-style="none" border-width="thin" extent="15mm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence initial-page-number="1" master-reference="folap">

    <fo:static-content flow-name="xsl-region-before">
      <fo:block font-size="9pt">Minta dokumentum -  egy hasábos</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body">
      <fo:block text-indent="1em" font-family="sans-serif" font-size="14pt"
      font-weight="bold" background-color="#EEEEEE" line-height="6mm"
      text-align="center">
        Szöveg.....
      </fo:block>
      <fo:block space-before="1em" padding-start="3mm" padding-end="3mm"
      font-family="sans-serif" font-size="12pt" text-align="justify">
        <fo:block>
          Szöveg.....
        </fo:block>
        <fo:block>
          Szöveg.....
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>

  <fo:page-sequence master-reference="lap2">
    <fo:static-content flow-name="xsl-region-before">
        <fo:block font-size="9pt">Minta dokumentum - két hasábos</fo:block>
    </fo:static-content>

    <fo:static-content flow-name="xsl-region-after">
      <fo:block text-align="center">
        kéthasábos lap
      </fo:block>
    </fo:static-content>
```

```
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="12pt" text-align="justify" hyphenate="true">
        <fo:block space-after="1em">
          Szöveg.....
        </fo:block>
        <fo:block space-after="1em">
          Szöveg.....
        </fo:block>
        <fo:block>
          Szöveg.....
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
```

5. Mintapélda: Vegyünk egy egylapos ismertetőt leíró dokumentumot, amelyben szerepelnek a fontosabbnak tekinthető alap formátum vezérlő utasítások, mint:

   - lap paraméterek beállítása,
   - régió paraméterek beállítása,
   - fejléc készítése,
   - lap sorszámozása,
   - margók beállítása,
   - szövegrész keretezése,
   - bekezdések szövegigazítása,
   - cím megadása,
   - bekezdések formátum megadása,
   - fontkészlet kijelölése,
   - táblázat készítése,
   - grafikus kép beszúrása.

   A minta XSL-FO állomány tartalma:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  >

<fo:layout-master-set>
  <fo:simple-page-master master-name="fosema"
    page-height="29.7cm"
    page-width="21cm"
    margin-left="2cm"
    margin-right="2cm"
```

```
      margin-top="2cm"
      margin-bottom="2cm"
      reference-orientation="0"
      writing-mode="lr-tb">

      <fo:region-body
        margin-bottom="1cm"
        margin-top="2cm"
       />
      <fo:region-before extent="1cm"   overflow="hidden"/>
      <fo:region-after extent="1cm"  border-width="thick"/>
      <fo:region-start extent="1cm" />
      <fo:region-end extent="1cm" />


   </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="fosema">
   <fo:static-content flow-name="xsl-region-before">
      <fo:block  text-align="right"
        font-size="10pt"
        color="grey"
        line-height="0.7cm"  margin="6pt">
         <fo:page-number/> . oldal
      </fo:block>
   </fo:static-content>
   <fo:flow flow-name="xsl-region-body" font-family="Courier">
      <fo:block text-align="center" font-size="14pt"    font-weight="bold"
       margin-bottom="2cm">
           Tisztelt Olvasó!</fo:block>
      <fo:block text-align="justify" font-size="12pt"  text-indent="0.5cm" >
        Ha módosítani szeretnénk a Fiat típusú autók árát az alábbi séma mellett,
        akkor elöbb  el kell jutnunk navigációs müveletekkel mindazon csomóponthoz,
        amely ár mezöt tartalmaz és a Fiat tipusú autó leiró csomóponthoz tartozik,
        majd ott egy újabb paranccsal átírhatjuk a csomóponthoz tartozó szövegértéket.
        Mivel a logikailag egybetartozó típuselem és az ár elem két különbözö elérési
        útvonalon  helyezkedik el, ezért nem lehet egyszerüen leírni a
        módosítást elvégzö tevékenység kódját sem.
      </fo:block>
      <fo:block  text-align="center"
        font-size="14pt"
         font-weight="bold"
        color="black"
        line-height="0.7cm" border-style="solid" border-width="thin"  margin="1cm">
        Figyelem! A vezeték érintése tilos és életveszélyes!

      </fo:block>
      <fo:block font-family="Arial">
        Most egy pár sor következik a táblázatokról. Az alábbi táblázatban megadjuk
        az egyes hónapokban elfogyasztott fagylalt gombócok darabszámát. Sajnos
        részletes statisztikák nem állnak rendelkezésre arról, milyen a gombócok
        fajta szerinti megosztása.
      </fo:block>

      <fo:table  border-style="solid" margin-bottom="1cm" margin-top="1cm">
         <fo:table-header>
           <fo:table-row>
             <fo:table-cell border-style="solid" ><fo:block>  </fo:block></fo:table-cell>
             <fo:table-cell border-style="solid" ><fo:block>január</fo:block></fo:table-cell>
             <fo:table-cell border-style="solid" ><fo:block>február</fo:block></fo:table-cell>
```

```
                <fo:table-cell border-style="solid" ><fo:block>március</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>április</fo:block></fo:table-cell>
              </fo:table-row>
            </fo:table-header>
            <fo:table-body>
              <fo:table-row>
                <fo:table-cell border-style="solid" ><fo:block>Miskolc</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>234</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>465</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>281</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>657</fo:block></fo:table-cell>
              </fo:table-row>
              <fo:table-row>
                <fo:table-cell border-style="solid" ><fo:block>Baja</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>24</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>146</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>71</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>257</fo:block></fo:table-cell>
              </fo:table-row>
              <fo:table-row>
                <fo:table-cell border-style="solid" ><fo:block>Szeged</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>534</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>765</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>881</fo:block></fo:table-cell>
                <fo:table-cell border-style="solid" ><fo:block>1257</fo:block></fo:table-cell>
              </fo:table-row>

            </fo:table-body>

          </fo:table>
          <fo:block  font-style="italic">Most még egy ábra következik:          </fo:block>
          <fo:block width="100%" text-align="center"> <fo:external-graphic
          src="H:\PROJEKTEK\TAMOP_JEGYZETEK\XML\sample4.jpg"  content-width="5cm"
          scaling="uniform"/></fo:block>
        </fo:flow>
      </fo:page-sequence>

</fo:root>
```

A generált PFD dokumentum megjelenítését az alábbi ábra adja meg:

Tisztelt Olvasó!

módosítani szeretnénk a Fiat típusú autók árát az al
mellett, akkor előbb el kell jutnunk navigációs műveletel
lazon csomóponthoz, amely ár mezőt tartalmaz és a Fiat tí
leíró csomóponthoz tartozik, majd ott egy újabb paranc
hatjuk a csomóponthoz tartozó szövegértéket. Mivel a logika
etartozó típuselem és az ár elem két különböző elérési útvon
ezkedik el, ezért nem lehet egyszerűen leírni a módosí
gző tevékenység kódját sem.

Figyelem! A vezeték érintése tilos és életveszélyes!

egy pár sor következik a táblázatokról. Az alábbi táblázatban megadjuk az egyes
pokban elfogyasztott fagylalt gombócok darabszámát. Sajnos részletes statisztik
állnak rendelkezésre arról, milyen a gombócok fajta szerinti megosztása.

	január	február	március	április
olc	234	465	281	657
	24	146	71	257
ed	534	765	881	1257

még egy ábra következik:

26. A generált PDF dokumentum

Irodalomjegyzék

[1] Tidwell Dough. *XSLT*. O'Reilly. 2001.

[2] Herpers F.J.- Sebestyen T. J.. *XSL*. Verlag moderne industrie Bucs AG . 2002.

[3] Holzner S.. *XSLt-Anwendung und Referenz*. Markt Technik Verlag. 2002.

[4] Bach M. *XSL und XPath*. Addison-Wesley Publisher. 2000.

[5] Deitel, Nieto, Lin, Sadhu. *XMl How to program*. Prentice Hall Publisher. 2001.

[6] Kranzler C.. *XML / XSL*. markt und Technik. 2002.

[7] Kay M. *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wrox. 2008.