Debreceni Egyetem
Informatika Kar

Felhasználói felületek fejlesztése XML alapokon

Témavezető:
Jeszenszky Péter
egyetemi tanársegéd

Készítette:
Szeverényi Attila
programozó matematikus

Debrecen
2008

Tartalomjegyzék

1. Bevezetés

1.1. A felhasználói felületekről

Jelen szakdolgozat témája olyan felhasználói felületeknek a készítése, amelyek XML alapokon nyugszanak, tehát megírásuk valamilyen olyan nyelven, vagy nyelv segítségével, eszközrendszerével történt, amely az XML-re épül.
A felhasználói felületekről röviden néhány mondatban [12]:
A felhasználói felületek (angol nevén User Interface, vagy rövidítve egyszerűen csak UI) létrehozásának fő célja nem más, mint a kommunikáció megvalósítása az ember és a gép között. Informatikai vonatkozásában ez gyakorlatilag nem jelent mást, mint a felhasználó és a számítógép közötti kommunikációt. A legegyszerűbb példa erre egy operációs rendszer kezelése, ott is ilyen felületek segítségével tudjuk megvalósítani az operációs rendszer különféle funkcióinak a vezérlését. Hogy egy egyszerű példát említsek, gondoljunk a Windows Vezérlőpultjára, ahol különböző beállításokat tudunk eszközölni, amelyek segítségével az operációs rendszerhez kapcsolódó értékeket tudunk megváltoztatni.
A felhasználó felületeknek hagyományosan három fajtája van:

- CLI (Command Line Interface): parancssoros felhasználói felület
- TUI (Text User Interface): szöveges felhasználói felület
- GUI (Graphics User Interface): grafikus felhasználói felület

Léteznek egyéb felhasználói felületek is, például hang segítségével történik a kommunikáció a hangvezérelt eszközöknél. Például: VoiceXML.
A CLI esetében a parancsok bevitele szöveges formában történik a billentyűzet segítségével, és az üzenetek is szöveges formában jelennek meg a monitoron. Erre egy jó példa a DOS, amely régen elterjedt operációs rendszer volt, és parancssorban adhattunk ki különféle utasításokat benne. A CLI-t ma is használják, gondoljunk a UNIX vagy a Linux parancssoros üzemmódjára. A TUI esetében úgynevezett karaktercellák vannak szöveges felirattal, és ezekkel a karaktercellákkal valósítunk meg például nyomógombokat és egyéb képi eszközöket. A Norton Commander, illetve más, hozzá hasonló programok jó példák a TUI-ra.
A GUI [13] esetén már vannak úgymond rendes grafikus elemek és szöveges elemek is. Sokkal változatosabb és esztétikusabb megjelenítést tesz lehetővé, mint a TUI. Erről szeretnék

egy kicsit több szót ejteni, mivel ezen dolgozatban is ilyen felhasználói felületek lesznek megvalósítva.

A GUI történetéről néhány szóban: A GUI-k előfutárának tekinthető felületet a Stanford Kutatóintézet dolgozói fejlesztették ki, és a saját On-line System nevű rendszerükben használták. Ezt az alapötletet fejlesztették tovább a Xerox PARC kutatóintézet munkatársai, és az általuk megalkotott felületet a Xerox Alto számítógépen kezdték el használni. Gyakorlatilag a legtöbb modern grafikus felületet ebből az eredetből szoktuk származtatni.

A grafikus felületek nagyfokú interaktivitás megvalósítására alkalmasak. Rajtuk keresztül tudunk kezelni operációs rendszereket, különféle felhasználói programokat. Természetesen ma már a számítógépen kívül egyéb eszközökön is elterjedt a használata (például: PDA-k, mobiltelefonok). Egy GUI grafikus, vizuális elemekből építkezik. Ilyenek például az ablakok, a menük, az ikonok, vagy különféle mutatóeszközök, mint például az egér vagy a laptopok érintőpadja. Az utóbbiakhoz kapcsolódik a képernyőn megjelenő mutató, vagy másnéven a kurzor, ez a mutatóeszközökkel való műveleteket (mozgás, kattintás) képezi le a képernyőre. A viszonylag könnyű kezelhetőség és átláthatóság következtében a kezdő, tapasztalatlanabb felhasználók is könyebben el tudnak boldogulni a különféle programok használatával, ha az rendelkezik grafikus felhasználói felülettel.

Hogyan is kapcsolódik össze a felhasználói felület és az XML? Az XML segítségével hordozható adat-leíró nyelveket tudunk létrehozni, természetesen ha az XML szabályrendszerét betartjuk. Az XML-ről és ilyen leíró nyelvekről a későbbiekben lesz szó. Általuk tudunk létrehozni felhasználói felületeket. Ezek bonyolultsága attól függ, hogy az adott nyelv mennyi mindent tud megvalósítani. A webes alkalmazásoknak a kliens oldalon (felhasználói oldalon) megjelenő felületének létrehozásában is segítségünkre vannak az ilyen nyelvek. A dolgozatban egy ilyen konkrét nyelv lesz részletesen bemutatva, az XForms, illetőleg más leíró nyelvek említés szintjén fognak szerepelni, tömören összefoglalva a lényegüket.

1.2. Felépítés

A dolgozat első részében az XML-ről lesz szó, mint az alapnyelvről, a kiindulási nyelvről. Ezt követően néhány leíró nyelvről fogok szót ejteni, egyrészt XML-en alapuló nyelvekről,

másrészt olyan nyelvről is, amit fontosnak tartok megemlíteni az XML kapcsán (például HTML).
Ezeket követően az XForms részletes bemutatása következik, szó lesz arról, hogy hogyan épül fel, milyen elemei vannak, 1-1 példával illusztrálva, majd pedig a legvégén három olyan példaprogram bemutatása következik, amelyek XFoms segítségével lettek megvalósítva. Látható lesz a forráskódjuk, illetve le lesz írva a működésük is.

2. Az XML [17], [30]

2.1. Az XML kialakulása

Az XML az Extensible Markup Language rövidítése, ami magyarra fordítva kiterjeszthető jelölőnyelvet jelent. A World Wide Web Consortium (röviden W3C) terméke és szabványa. Az SGML-ből származtatható. A HTML nyelv is az SGML-ből származik, de ennek ellenére az XML és a HTML alapvetően különbözik egymástól, mert például a HTML-ben előre definiált elemekkel tudunk dolgozni, míg az XML ezzel szemben egy olyan mechanizmust biztosít a számunkra, amely segítségével a dokumentumban használható címkekészleteket lehet létrehozni.
Maga az SGML (Standard Generalized Markup Language, magyarul standard általános jelölőnyelv) még a ’80-as években jelent meg és lett szabvány. Az SGML-szabvány elég bonyolult, ami azt jelenti, hogy például egy dokumentum esetében nehéz ellenőrizni azt, hogy megfelel-e a szabványban foglaltaknak. A gyakorlatban is használták, de később jó kiindulási alapot szolgáltatott más nyelvek számára.
A HTML ’91-ben jött létre, de egy idő után nyilvánvalóvá vált, hogy a bonyolultabb adatok kezelése és tárolása nehézkes benne, mivel a HTML inkább azt a célt szolgálja, hogy megjelenítsük vele az adatokat. Így vált szükségessé egy olyan jelölőnyelv létrehozása, amelyik megoldásként szolgál erre a problémára. A W3C csapata alkotta meg ezt a nyelvet XML néven. Maga a fejlesztőcsapat 1996-ban alakult, és 1998-ban vált hivatalos ajánlássá az XML. Elég sikeres nyelvnek bizonyult. Ebben nagy szerepet játszik egyrészt az, hogy könnyen használható, az SGML-nek egy kisebb, egyszerűbb és szigorúbb szabályok mentén felépülő részhalmaza, ezért könnyebben megtanulható és használható, és egyszerűbb az

XML-alapú programok írása. Másrészt a sikerében szintén nagy szerepe volt annak, hogy platformfüggetlen, tehát hordozható, egy Macintoshon ugyanúgy meg tudjuk jeleníteni, mint a saját számítógépünkön, ami esetünkben egy PC.
Az XML fő célja az információcsere segítése, legyen szó akár webes, akár nem webes információcseréről.
Manapság a legelterjedtebb XML verzió az XML 1.0, ennek a harmadik kiadása van érvényben, én is ezt vettem alapul. Létezik az 1.1-es verzió is, bár tudomásom szerint ez nem annyira elterjedt.
A következőkben vegyük sorra azt a 10 pontot, amit fejlesztői a nyelv megalkotásakor célként kitűztek.

2.2. A 10 pont [30]

- Az XML legyen egyenesen az Interneten keresztül használható.
- Az XML lehetőleg minél szélesebb körét támogassa az alkalmazásoknak.
- Az XML legyen kompatibilis az SGML-lel.
- Könnyen lehessen XML dokumentumokat feldolgozó programot írni.
- Az XML opcionális jellemzőinek, lehetőségeinek a száma minél minimálisabb legyen, ideális esetben nulla.
- Az XML dokumentumok legyenek az emberek számára is könnyen olvashatóak és észszerűen áttekinthetőek.
- Az XML szabvány legyen minél gyorsabban elkészíthető.
- Az XML szabványa legyen formális és tömör.
- Könnyen létre lehessen hozni XML dokumentumokat.
- Az XML jelölésrendszerében a tömörségnek csekély jelentősége van.

Ezek voltak a W3C fejlesztőcsapatának fő célkitűzései. Esetleg el lehet gondolkozni azon így tíz év távlatából, hogy vajon sikerült-e a célkitűzéseiknek eleget tenniük.

2.3. Az XML-ről általánosságban

Mint már említettem, a HTML kész elemekből építkezik, míg az XML csak egy mechanizmust biztosít a számunkra, amely segítségével a dokumentumban használt elemeket

tudjuk létrehozni. Az XML nem a HTML helyett, nem annak leváltására jött létre. Az XML-t használhatjuk például a HTML kisegítésére (például weboldalhoz tartozó információk tárolása), így bővítve ki jelentős mértékben a weblapok lehetőségeit.

Az XML dokumentumok értelmezését, feldolgozását az XML feldolgozó végzi. A böngészők beépített modul segítségével tudják megjeleníteni ezeket a dokumentumokat. Például a Microsoft Internet Explorerjében egy külön szoftvermodul végzi ezt. Ennek neve a Microsoft XML Core Services, rövidebb nevén MSXML. A böngészőben ez alapból benne van, így ha feltelepítjük, akkor az már képes egy XML dokumentum értelmezésére és megjelenítésére. Több verziója létezik, az én gépemen az MSXML 6.0 van feltelepítve. Más böngészőkben is hasonló módon működik ez a dolog.

Maga a megjelenítés alapesetben úgy történik, hogy a böngészőben megjelenik a dokumentumfa. Ebben a dokumentumfában lépkedhetünk, a fa egyes ágait ki tudjuk nyitni, illetve be tudjuk zárni. Ha megadunk valamilyen stíluslapot is, akkor a stíluslapnak megfelelően jelenik meg az XML dokumentum. Ilyenkor a különböző feldolgozási utasítások, a megjegyzések, az elemek nyitó- és záró címkéi, az elemek tulajdonságai nem jelenítődnek meg, csak az elemtartalmak. A stíluslapon található szabályok is az elemek tartalmára vonatkoznak.

Két fő fogalmat mindenképpen meg kell említeni az XML dokumentumokkal kapcsolatban:

- **Jól formázottság**
- **Érvényesség**

Jól formázottság: Minden olyan dokumentumot jól formázottnak nevezünk, amelyik az XML-szabványban lefektetett szintaktikai szabályok szerint épül fel.

A következőkben arról szeretnék beszélni, hogy egy jólformált XML dokumentumban hogyan néz ki a szintaktikai felépítés, milyen főbb szintaktikai szabályok vannak

Az XML egyik fontos alapfogalma a **név**. Mivel valamilyen szinten köthető a szintaktikai szabályokhoz, így itt tisztáznám a fogalmát. Lehetnek nevei az elemeknek, egyes elemek tulajdonságainak, stb. A név egy olyan karaktersorozat, amely tetszőleges hosszúságú lehet, mindig betűvel, aláhúzásjellel vagy kettősponttal kell kezdődnie, amit betű, szám, pont, kettőspont, kötő- illetve aláhúzásjel követhet. Név viszont nem kezdődhet az „xml” karaktersorozattal, és ennek bármilyen kis- és nagybetűs formájával, mivel az foglalt szónak

számít. Itt említeném meg, hogy az XML különbséget tesz a kisbetűk és a nagybetűk között, erre figyeljünk.

Az XML dokumentumokban Unicode karaktereket használhatunk. Az Unicode gyakorlatilag tartalmazza a világ összes nyelvének a karaktereit.

És most lássunk néhány fontosabb megszorítást (szintaktikai szabályt) a jól formázottsághoz kapcsolódóan:

- a dokumentum csakis egy legfelső szintű elemmel rendelkezhet, ezt nevezzük gyökérelemnek vagy dokumentumelemnek. Ezt az elemet megjegyzések vagy a feldolgozónak szóló utasítások előzhetik csak meg.
- a címkék határolják az elemeket, ezek alakja a következő: `<név>` , illetve `</név>`
- egy nemüres elemet mindig egy nyitó és egy záró címkével kell határolnunk az alábbi módon: nyitó címke: `<név>`, záró címke: `</név>`
- ha üres az elem, akkor nem szükséges külön záró címkét kiraknunk, helyette írhatjuk ezt is: `<név/>`
- a kezdő és záró címkében lévő neveknek természetesen meg kell egyezniük, különösen figyelve a kis- és nagybetűkre
- megadhatunk XML fejlécet is, bár ez nem kötelező, de az XML-szabvány előírja. Hogyha van fejléc, akkor az mindig a legelső sorban van, például az alábbi formában:

  ```
  <?xml version="1.0" encoding="ISO-8859-2" ?>.
  ```

  Itt látható, hogy megadhatjuk a verziót, a kódolást. Az XML feldolgozó alapesetben UTF-8 vagy UTF-16 kódolást feltételez, a fenti példában a Latin-2 karakterkódolás van megadva.
- egy elemnévnek mindig meg kell felelnie a név fent leírt definíciójának
- a „&” és a „<” karaktert literális formában sehol sem lehet használni, sem szövegben, sem attribútumértékben. Ilyen formában kizárólag jelölőhatárolóként, vagy megjegyzésben, feldolgozási utasításban, CDATA-részben fordulhatnak elő. Más helyeken karakterhivatkozással, vagy entitáshivatkozással (& és <) használható.

  Például: `<ember neve="’ Kiss Béla&amp;1" >`. Itt a határolókarakter a ” , így a ’ szabadon használható belül, illetve az `&amp;` pedig egy entitáshivatkozás, ezért használható benne az &.

- az egyes elemeknek megadhatunk különféle tulajdonságokat is, amelyekből több is lehet.

  Formálisan így néz ki egy ilyen megadás: `<név tulajdonság="érték">` . Ez egy elem nyitórészében szerepel. Az érték megadható mind dupla (”), mind pedig szimpla (’) idézőjelek között, az érték tartalmazhat bármilyen karaktert, leszámítva a határolókaraktereket és az „&” illetve „<” karaktert (ld. előző pont). Egy nyitócímkén belül egy adott nevű tulajdonság csak egyszer adható meg jólformált XML dokumentumban.
- az elemeket tetszőlegesen ágyazhatjuk egymásba. Azt az elemet, amelyik tartalmazza a másik elemet, szülőelemnek nevezzük, míg a tartalmazott elemet gyerekelemnek. Itt a legfontosabb szabály az, hogy az elemek nem fedhetik át egymást.

  Erre egy példa: `<ember><név></név></ember>` , ez így helyes. Ez pedig helytelen: `<ember><név></ember></név>` , de például a HTML-ben ez megengedett volt.
- az elemek szöveges tartalma tetszőleges hosszúságú karakterlánc lehet. A „&” és a „<” karakter használatáról már feljebb szóltam, az érvényes itt is.
- megadhatunk feldolgozási utasításokat is, amelyeket az XML dokumentumban bárhol elhelyezhetünk, kivéve más jelöléseken belülre. Egy ilyen utasításban megadunk egy célszoftvert és más információkat, amelyeket mondhatunk akár paramétereknek is, a végén a példában világosabb lesz. A feldolgozási utasítás lehet olyan, amit egy általunk létrehozott szoftver tud értelmezni, de lehet szabványos is, tehát olyan, hogy az XML feldolgozónak tudnia kell értelmezni. Például: `<?xml-stylesheet type="text/css" href="xy.css" ?>`. Itt láthatjuk azt, hogy a „paraméterek” megmondják, hogy egy CSS stíluslapról van szó, illetve megadják annak a forrását is.
- minden olyan egyednek – amelyre hivatkozás történik valahol –, szintén jól formázottnak kell lennie. Itt ejtenék néhány szót az egyedekről: egy XML dokumentum egy vagy több tárolási egységből állhat, ezeket nevezzük egyedeknek, mindegyiknek van tartalma, és az azonosításukra a nevük szolgál (kivéve a dokumentumegyedet és a külső DTD-alkészletet). Az egyedek lehetnek elemzett vagy nem elemzett egyedek, általános vagy paraméteregyedek, külső vagy belső egyedek. Az elemzett egyed tartalma helyettesítő szöveg, az egyedre történő

hivatkozás ezen szöveg beillesztését jelenti a dokumentumba. A nem elemzett egyed tartalma bármi lehet. Az általános egyedekre a dokumentumon belül tudunk hivatkozni egyedhivatkozással, míg a paraméteregyedeket a DTD-ben tudjuk használni paraméteregyed-hivatkozással. Belső egyedeknél az egyeddeklarációban adott az egyed tartalma, a helyettesítő szöveg. Belső egyed csak elemzett egyed lehet. Minden olyan egyed, amelyik nem belső egyed, az külső egyed. A külső egyed lehet elemzett és nem elemezett is.

- CDATA elem: erről annyit érdemes tudni, hogy az ilyen részben leírt szöveget a feldolgozó sima szövegként értelmezi. Itt használhatunk olyan karaktereket is „büntetlenül", amelyekről korábban említettem, hogy máshol nem, vagy csak speciális esetben használhatók. Egy CDATA rész formálisan így néz ki:
  `<![CDATA[ bármilyen szöveg ]]>`

Ezek a jól formázottsággal kapcsolatos megszorítások nem törekednek a teljességre, igyekeztem inkább a véleményem szerint fontosabbakat kiragadni.
Nem megszorítás, de érdemes megemlíteni a megjegyzéseket is. Ezek olyan részek, amelyeket a feldolgozó figyelmen kívül hagy. Ezek egyrészt nekünk szólnak, másrészt azoknak, akik olvassák a dokumentumot. A megjegyzéseket `<!--` és `-->` jelek közé írjuk.
A beépített XML feldolgozó modulok végeznek a megjelenítés előtt jól formázottsági ellenőrzést, és ha a dokumentum nem felel meg a szabályoknak, akkor nem jelenik meg, helyette egy hibaüzenettel találjuk szembe magunkat, ahol látjuk kiírva, hogy mi a hiba, és az hol található a dokumentumban. Én Internet Explorert, illetve Mozilla Firefoxot használtam a munkám során. Utóbbiban annyival több információt látunk, hogy nem csak a hibás sort látjuk kiírva, hanem azt is, hogy hányadik sor hányadik karakterénél van a hiba. Ez egy nagyobb terjedelmű dokumentum esetén lehet hasznos, mert így kevesebbet kell keresgélnünk.

Érvényesség: Alapvetően mikor érvényes egy XML dokumentum? Akkor, ha egyrészt jól formázott. Másrészt felépítésében és tartalmában is meg kell felelnie a rá vonatkozó szabályoknak, mely szabályokat magában a dokumentumban is megadhatjuk, de megadhatjuk őket egy külső fájlban is, amelyre a dokumentumunkban hivatkoznunk kell. Ilyen szabályok megadására szolgál a **DTD** (Document Type Definition, magyarul dokumentumtípus

definíció), vagy az **XML-séma** (XML Schema). A DTD tartalmazza vagy jelzi mindazokat a jelölődeklarációkat, amelyek a dokumentumosztályok számára adnak meg egy nyelvtant, vagyis nyelvtani szabályok összeségét. A dokumentumtípus-deklaráció hivatkozhat egy külső részhalmazra, amely tartalmazza a jelölődeklarációkat, de belső részhalmazként is tartalmazhatja ezeket a deklarációkat. Természetesen lehet egyszerre külső és belső részhalmaz is, ilyenkor a dokumentum DTD-jét a kettő együttesen fogja alkotni. Az XML-séma definíció a DTD-vel szemben csakis külső fájlban lehet.

Fontos megemlíteni, hogy a böngészők elvégzik a dokumentumon a jól formázottságra vonatkozó ellenőrzést, de az érvényességre vonatkozót már nem. Erről külön nekünk kell gondoskodni, például egy megfelelő script segítségével.

Nem szeretnék egyik módszerbe sem mélyen beleásni, csak röviden bemutatnám őket.

- **DTD** [30]**:** elhelyezhető mind a dokumentumon belül, mind azon kívül is. Része az XML-szabványnak, viszont ennek ellenére nem XML szintaxissal rendelkezik, mivel az SGML-ből ered. Mint említettem, a böngésző csak a jól formázottságra végez ellenőrzést, és ha DTD-t is megadtunk a dokumentumban, akkor ezt az ellenőrzést azon is elvégzi. A dokumentumtípus deklaráció megadja magát a DTD-t, azt, hogy hol van. Megadása az XML dokumentum fejlécében történik a következőképpen:

  ```
  <!DOCTYPE dokumentumelem_név [jelölődeklarációk]>
  ```

  Külső DTD deklaráció megadása, ahol az URL általában .dtd-re végződik:

  ```
  <!DOCTYPE dokumentumelem_név SYSTEM "url">
  ```

  Többféle deklarációra van lehetőségünk. Deklarálhatunk elemtípusokat, ennek segítségével egyes elemekre vonatkozó megszorításokat adhatunk meg. Például:

  ```
  <!ELEMENT vezetéknév (#PCDATA)>
  ```

  Lehetőségünk van bizonyos jellemzők deklarálására. Például:

  ```
  <!ATTLIST személy név CDATA #REQUIRED>
  ```

  Deklarálhatunk egyedeket is, az egyedekről röviden már tettem fentebb említést. Egy példa egyed létrehozására:

  ```
  <!ENTITY születési_hely "Debrecen">
  ```

  A jelölődeklaráció tehát lehet elemtípus-deklaráció, lehet attribútumlista-deklaráció, lehet egyeddeklaráció, és lehet jelölésdeklaráció. Lehetőség van arra is, hogy egyidejűleg használjunk külső és belső DTD-t, és ha mindkettő van, akkor a belső DTD élvez elsőbbséget a külsővel szemben. A DTD-hez kapcsolódóan említeném

meg a **névterek**et [31]**.** Minden elemtípus csak egyszer deklarálható a DTD-ben. Azt mondhatjuk, hogy egy elemet csak egyféleképpen lehet deklarálni, tehát nem lehet ugyanazt az elemet különböző módon megadni. A névterekkel ez a probléma kiküszöbölhető, minősített neveket használunk, amelyek segítségével a névütközési problémákat el tudjuk kerülni. A minősített nevek lehetnek előtag nélküliek, de rendelkezhetnek előtaggal is. Maga a névtér az XML dokumentumokban elemek és tulajdonságok neveként használt neveknek az összesége, amelyeket egy abszolút URI-hivatkozás azonosít. Az URI helyén nem állhat üres karakterlánc, illetve nem fontos, hogy ez az URI egy ténylegesen létező erőforrásra mutasson. A minősített nevek előtagjához a névtér-deklarációban adjuk meg a névtér-nevet. A névtér megadása a dokumentumban speciális tulajdonsággal történik, méghozzá kétféle módon:

1. Alapértelmezett névtér-deklarációval, amikor a tulajdonság neve `xmlns`. Ez a dokumentumban a minősített nevű és előtaggal nem rendelkező elemekre vonatkozik. A minősített nevű és előtaggal nem rendelkező tulajdonságokra ez nem áll, mivel ezek nem tartoznak egyik névtérbe sem. Előfordulhat, hogy névtér-névnek üres karakterlánc van megadva, ilyenkor a névtér hatáskörében található előtag nélküli, minősített nevek nem fognak egyik névtérbe sem beletartozni.

2. A másik megadási mód esetén a tulajdonság neve `xmlns:előtag` alakú, ahol az előtag betűvel vagy aláhúzásjellel kezdődő, és kettőspontot nem tartalmazó név. Egy névtér-deklaráció hatásköre az az elem, amely nyitócímkéjében a névtér-deklaráció szerepel. Egy egyszerű példa a névterekre:

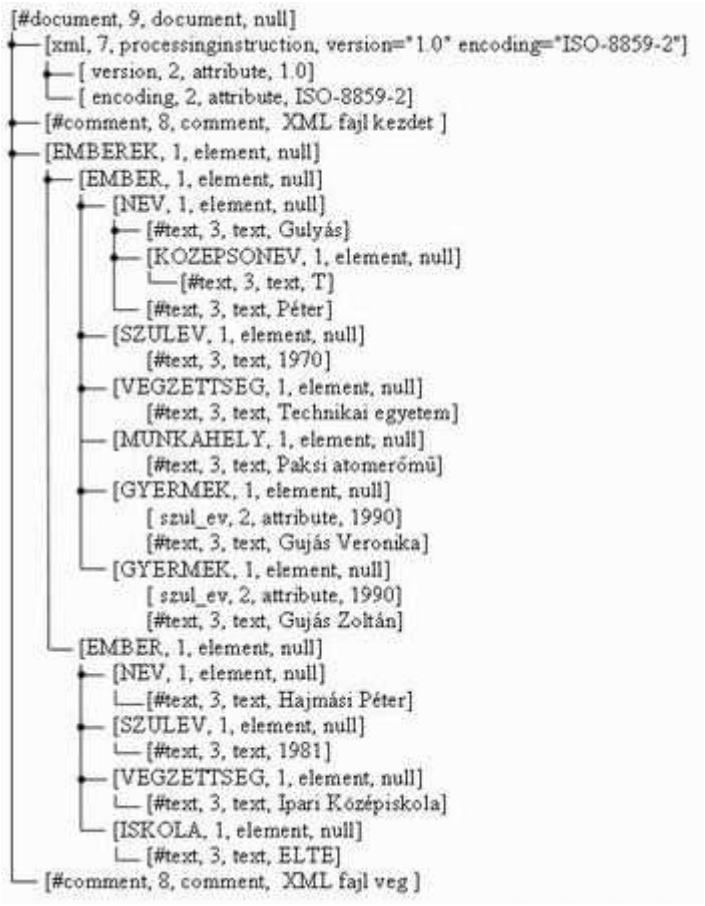
```
<EMBEREK xmlns:beosztott="http://www.xykft.hu/beosztott"
xmlns:fonok="http://www.xykft.hu/fonok"> …
<fonok:EMBER><fonok:NÉV>Kiss József</fonok:NÉV></fonok:EMBER>
<beosztott:EMBER><beosztott:NÉV>Mező
János</beosztott:NÉV></beosztott:EMBER>
```

- **XML-séma** [17]**:** A W3C által létrehozott nyelv. Az XML-séma dokumentumot csak külső fájlban adhatjuk meg, amelynek a végződése általában .xsd. Ez az XML Schema Definition rövidítéséből jön. Ellentétben a DTD-vel, itt a szintaktika már nem különbözik, megegyezik az XML szintaxisával. Több lehetőséget kínál, mint a DTD. Az XSD fájloknak is ugyanúgy jól formázottnak kell lenniük. Meg kell adnunk a `"http://www.w3.org/2001/XMLSchema"` névteret, ezzel is jelezve, hogy

XML-sémáról van szó. Lehetőségünk van egyszerű, illetve összetett elemtípusok deklarációjára, illetve tudunk attribútumokat, magyarul jellemzőket is deklarálni. Az XSD fájlt nem kell külön belinkelni az XML dokumentumba. Egy XML dokumentum egy megadott XML-séma szerinti ellenőrzését egy ellenőrző szoftver végzi. Ha mondjuk más séma alapján akarjuk az XML dokumentumot ellenőrizni, akkor azt könnyen megtehetjük, az ellenőrző szoftver számára egy másik sémafájlt adunk meg. Külső DTD esetén ugyanezt úgy tudjuk elérni, hogy az XML dokumentumba nyúlunk bele, és ott módosítjuk a nevet.

2.4. Az XML felhasználása

Az XML egyik fő felhasználási területe az adatok tárolása. Teljes egészében szöveges alapú, éppen ezért könnyen és világosan áttekinthető, és feldolgozható. Azt is mondhatjuk, hogy egy egyszerűbb adatbázis szerepét képes ellátni.
Az XML-ben a tartalom és a megjelenítés különválik, így külön-külön hatékonyabban tudjuk kezelni őket.
Egy XML dokumentum felépítésére a faszerkezetű struktúra a jellemző. Ezzel hatékony feldolgozást tudunk elérni. Mind HTML, mind XML oldalak feldolgozása megvalósítható a **DOM** (dokumentum objektum modell) [21] segítségével (létezik még például a STAX API is, ami a soronkénti feldolgozással analóg). Itt most a HTML-es verzióra nem térek ki, bár példaprogramjaim megírásában ezt is felhasználtam kis mértékben. A DOM W3C szabvány, amely minden dokumentumelemet objektumok formájában hierarchikusan csoportosít, és metódusokat biztosít ezek eléréséhez. Az XML DOM röviden nem más, mint egy olyan szabvány, amely megmondja, hogy hogyan tudunk a dokumentumbeli XML elemekkel dolgozni, legyen szó akár egy új elem hozzáadásáról, vagy egy régi módosításáról vagy törléséről. És most tekintsük át a fastruktúrát! A fában a gyökérelem a `document` elem, ennek vannak leszármazottjai, amelyek csomópontként vannak letárolva a fában. Ezek lehetnek feldolgozó utasítások, elemek, megjegyzések, stb. Mindegyik csomópontot többféle jellemzővel írhatjuk le, ezek megtalálhatóak a DOM szabványában. Néhány fontosabb jellemző: a csomópont neve, a csomópont típusa számmal, a csomópont értéke. Ezek mind külön-külön elérhetők és felhasználhatók egy dokumentum feldolgozásakor. Lássunk erre egy képes példát a jobb érthetőség kedvéért (forrás: http://www.prog.hu):



1. ábra: DOM

Látható, hogy a legelső gyerekelem mindig a fejléc. Láthatjuk, hogy az egyes elemeket `element` típusú gyerekelemben tároljuk, amelyeknek mindenhol `null` az értéke. Azért van ez így, mert a személynevet, végzettséget, iskolát, stb. külön szöveges tartalomként kezeli, ezért is kerültek ezek külön csomópontba. És mivel ezeknek nincs külön nevük, ezért a fában névként a típus nevét kapják, előttük egy kettőskereszttel. Egy-egy ember gyermekét megjelenítő csomópontban pedig az is észrevehető, hogy az attribútum típusok is külön csomópontba kerülnek. Ugyebár ezek azok, amiket a nyitócímkében tulajdonságként megadhatunk. Különféle műveletekkel tudunk lépkedni a fában és elérni a különböző csomópontokat és azok tartalmát. Ha például egy HTML oldalon egy XML dokumentumot dolgozunk fel, akkor ezek nagy segítségünkre lehetnek. Ezekre nem térnék ki részletesen, a példaprogramokban előfordul egy-két ilyen művelet a HTML DOM-mal karöltve. De hogy mégis lássuk, miről is van itt szó, íme egy-két ilyen metódus:

- `NodeList Node.childNodes`: egy adott csomópont gyerekelemeinek a listájával tér vissza

- `String Node.nodeValue`: ez egy csomópont értékével tér vissza
- `NodeList Element.getElementsByTagName(String tagName)`: a megadott elem gyerekelemei közül azokat adja vissza egy listában, amelyeknek az elemneve megegyezik a paraméterül kapott névvel

Megemlítenék még egy másik technikát, amellyel XML dokumentumokat tudunk feldolgozni, megjeleníteni. Ez az **adatkötéses technika**, ami a **DSO**-n (Document Source Object) alapul. Lényege az, hogy a DSO segítségével jelenítjük meg az XML dokumentumbeli adatokat a HTML oldalon, méghozzá úgy, hogy azokat egyes HTML elemekhez kötjük hozzá. Mire is jó ez a technika? Gondoljunk csak bele, ha nekünk szükségünk van valamilyen adatra, akkor azokat mindig újra és újra le kellene tölteni a szerverről, így mindig terhelve azt. Ráadásul nem csak mi használunk egy szervert, hanem rajtunk kívül még sok más ember, akiknek mondjuk pont ugyanazokra az adatokra van szüksége, mint nekünk. Ez jelentősen lelassíthatja a letöltést, hiszen a szervernek mindenki számára ugyanazokat az adatokat kell elküldenie. Ez a technika viszont az adatok helyi, kliensoldalon történő tárolását teszi lehetővé, így csökkentve a szerver leterheltségét. Ez úgy néz ki, hogy a kérésünkre válaszul a szerver legenerálja a kívánt adatokat valamilyen formátumban (ez a mostani esetben egy XML dokumentumot jelent), azt elküldi nekünk, és utána ezekkel az adatokkal már a helyi gépen dolgozunk. Ezt a feldolgozást segíti elő a DSO. Ezen technika részletezésébe nem mennék bele, csak egy nagyon egyszerű példát írnék. Íme két kiragadott sor egy HTML oldal forrásából:

```
…<XML ID="dso" SRC="munkas.xml"></XML>
…Név: <SPAN DATASRC="dso" DATAFLD="nev"></SPAN>…
```

Látható, hogy megadunk egy azonosítót, amivel tudunk hivatkozni az XML fájlra, aztán pedig meg kell adni a fájl nevét. A kód későbbi részében pedig egy `<SPAN>` elemhez kötjük ezt a fájlt, és hivatkozunk a `nev` nevű elemre. Mivel a `<SPAN>` egy darab elem megjelenítésére képes, így itt csak a legelső ilyen nevű elemnek a tartalma fog megjelenítődni, tehát a legelső munkás neve. De lehetőségünk van több elem megjelenítésére is, például táblázatok segítségével.

2.5. Egy XML dokumentum megjelenítése stíluslapokkal

Végezetül ejtsünk szót arról, hogy hogyan tudunk megjeleníteni egy XML dokumentumot stíluslapokkal. A DOM és a DSO is egyfajta megjelenítési módszer, viszont én az itt következő két módszert különvettem. Ezek a **CSS**, illetve az **XSLT**. Ezekről pár mondatban szeretnék csak írni.

- **CSS** [17]**:** Cascading Style Sheets, W3C szabvány, amely egy stíluslapnyelvet definiál XML és HTML dokumentumok megjelenítésére. A CSS stíluslap egy különálló dokumentumot jelent. Egy ilyen stíluslapot az alábbi kóddal tudunk hozzárendelni egy XML dokumentumhoz:

  ```
  <?xml-stylesheet type="text/css" href="stiluslap.css"?>
  ```

  HTML és XHTML esetén lehetőségünk van a dokumentumon belül is megadni stíluslap-információkat a `<head>` részen belül, vagy maguknál az elemeknél egy `style` nevű tulajdonság segítségével. A CSS nyelv külön szintaktikával rendelkezik, az egyes elemekre lebontva adhatunk meg különféle megszorításokat a megjelenítésükre vonatkozóan, például betűtípus, betűméret, háttérszín, beállíthatjuk a margótól való távolságot, stb. Részletekbe nem mennék bele, a W3C honlapján megtalálhatjuk a részletes leírását. Az XML esetén annyit viszont megjegyeznék, hogy a W3C a CSS helyett inkább az XSLT használatát javasolja.
- **XSLT** [26]**:** Extensible StyleSheet Language Transformation, kiterjeszthető stíluslap-nyelv transzformációk. W3C szabvány, .xsl végű fájlban szokás megadni a leírásokat. Egy XML dokumentumban az XSLT-re vonatkozó feldolgozási utasítás majdnem ugyanaz, mint a CSS esetén, csak `"type=text/xsl"`-t kell megadnunk benne `"type=text/css"` helyett. Az XSLT arra szolgál, hogy az XML dokumentumokat más formátumra tudjuk konvertálni, ami lehet például HTML, XHTML. Az XSLT feldolgozó alapesetben két inputfájlt olvas be, egy XML forrásdokumentumot, és egy XSLT stíluslapot, és létrehoz egy kimeneti dokumentumot. XSLT-n belül van lehetőség CSS, sőt, HTML kódot is használni. Így jóval bonyolultabb és részletesebb stíluslapok állíthatók elő. Minden egyes XML elemhez megadhatunk valamilyen sablont. Az XSLT feldolgozó végigjárja a dokumentumot mint egy fa adatszerkezetet, és minden elemre megnézi, hogy van-e hozzá sablon, ha van, akkor azt alkalmazza, ha nincs, akkor valamilyen beépített

sablont használ. Beépített sablon szabály például: `text` típusú csomópont esetén csak a szöveg jelenik meg, míg megjegyzés vagy attribútum esetén nem jelenik meg semmi. Egy XSLT dokumentumban mindig meg kell adni a „standard” XML fejlécet, illetve az XSLT névterét is, amihez az alábbi URI-t kell hozzárendelnünk: `"http://www.w3.org/1999/XSL/Transform"`. A névtér megadása kötelező, a névtér előtag általában az, hogy `xsl`, és a dokumentum további részében ezt minden XSLT elemnél meg kell adni, A verziószám megadása szintén kötelező. További részletekbe itt sem mennék bele, a W3C oldalain erről is találunk részletes leírást.

3. Felületleíró nyelvek

Az alábbiakban néhány XML-hez kapcsolódó felhasználói felületet leíró nyelvről lesz szó röviden. Legelőször viszont megemlítem a HTML nyelvet, mivel munkáim során ezt a nyelvet is felhasználtam, pontosabban az XHTML-t, amely nyelv köthető mind az XML-hez, mind a HTML-hez.

3.1. HTML [1]

A HTML a HyperText Markup Language rövidítése. Hasonlóan az XML-hez, ennek a nyelvnek is az SGML az őse. Jelenlegi verziója a 4.01-es, bár tudomásom szerint már létezik az 5-ös verziója is, vagy legalábbis fejlesztés alatt áll. Az 5-ös verzióban azt szeretnék elérni a fejlesztők, hogy kompatibilis legyen a 4.01-es verzióval, illetve az XHTML 1.0-ás, 1.1-es verzióival. Tágabb korlátokat enged meg egy kód megírása során, mint az XML, nincs benne annyi megkötés. Elég csak arra gondolni, hogy nem minden elemnél kötelező megadni záró címkét, sőt, van olyan elem is, amelyiknél tilos. Vagy például nincs különbség a kis- és nagybetűk között. Előre definiált elemekkel dolgozik, nem olyan érzékeny a hibákra, mint az XML. Ezt mint hátrányt említem, hiszen egy hibás HTML kódot is megjelenít egy böngésző, míg XML-nél vagy XHTML-nél ez nem lehet, ott hibaüzenetet kapunk.
Az SGML egy bonyolultnak nevezhető szabvány, ezt a HTML úgy oldotta meg, hogy a strukturáló és szemantikai elemeknek csak egy kisebb halmazát vette bele a saját szabványába, így viszonylag egyszerű dokumentumokat lehetett létrehozni. A HTML rövid

idő alatt nagy népszerűségre tett szert, világszerte gyorsan elterjedt, és az idők folyamán újabb és újabb elemekkel bővült az eszközkészlete. Sőt az is előfordult, hogy az újabb szabványban már nem ajánlották vagy egyenesen tiltották bizonyos korábbi elemek használatát. Sajnos ezek következtében a hordozhatóság terén egyre inkább felléptek kompatibilitási problémák.

3.2. XHTML

Extensible HyperText Markup Language rövidítése [14], [15]. Divatosan mondva nem más, mint a HTML megújulása XML alapokon. Mivel ez a nyelv elsősorban XML alapú, így több előnnyel is rendelkezik a HTML-hez képest, például:

- mivel az XML egyszerűbb nyelv a HTML-hez képest, így egy XHTML dokumentumot is hamarabb fel lehet dolgozni, mint egy HTML-t
- fentebb említettem, hogy egy HTML dokumentum akkor is megjelenik a böngészőben, ha történetesen vannak benne hibák. Ezzel szemben egy XHTML-alapú oldal csak akkor jelenik meg, ha hibáktól mentes, különben az XML-hez hasonlóan hibaüzenetet kapunk.
- egy XHTML oldalon belül bármilyen XML-re épülő nyelvet tudunk használni, például SVG-t vagy XForms-ot is, feltéve ha a megfelelő XHTML verziót használjuk

Az XHTML olyan dokumentumok nyelvéül lett szánva, amelyek olvashatóak XML-lel kompatibilis böngészőkkel, de bizonyos irányelvek betartásával olyan böngészőkkel is, amelyek a HTML 4-el kompatibilisek.
Jelenleg az XHTML-nek három specifikációja létezik. Az XHTML 1.0 a HTML 4.01-nek az XML-alapú megvalósítása, tehát gyakorlatilag ugyanolyan weboldalakat tudunk vele készíteni. A különbség abban van, hogy az XHTML-nek szigorúbbak a szabályai. Létezik három különböző dokumentumtípus hozzá, ezeket a fejlécben kell megadni, egy dokumentumtípus deklarációban. Ezek a `strict`, a `transitional` és a `frameset`. Az XHTML 1.1 az 1.0 `strict` továbbfejlesztése, itt a kinézetet csak stíluslappal tudjuk beállítani, illetve ez már modulokra van osztva. A harmadik specifikáció az XHTML 2.0. Ez gyakorlatilag még fejlesztési fázisban van, a HTML 4.01 és az XHTML 1.0 és 1.1

továbbfejlesztésének szánják, viszont egy teljesen új koncepció jellemző rá, így nem lesz felülről kompatibilis azokkal.

És most lássunk néhány megszorítást, ami az XHTML-re nézve kötelező, de a HTML-ben nem volt az, vagy másképp is megengedte:

- az XHTML-ben a HTML elemek és attribútumok nevét mindig kisbetűvel kell írni
- XHTML-ben az elemek kezdő és záró címkéjét is kötelezően meg kell adni, míg a HTML-ben ez nem mindig volt kötelező. Ha az elem üres, akkor egy kicsit szabadabbak a lehetőségeink, ilyenkor többféle módon is le lehet zárni, ezek mind helyesek: `<hr/>`, `<hr />`, `<hr></hr>` .
- az attribútumok értékeit mindig idézőjelek között kell szerepeltetnünk, függetlenül attól, hogy az érték szöveges vagy numerikus
- az attribútumok megadását nem lehet lerövidíteni, tehát itt nem lehet olyat írni például a `checked="checked"` helyett, hogy `checked`
- ahol a HTML-ben az elemek azonosítására `name` attribútumot használtunk, ott az XHTML-ben `id`-t kell használni
- a jól formázottság ugyanúgy érvényes az XHTML-re, mint az XML-re, például itt sem lehet szabálytalanul egymásba ágyazni az elemeket
- a `<html>`, `<head>`, `<body>` elemeket kötelező megadni
- a & és < elemeket kötelező entitással helyettesíteni

Ezek a főbb megszorítások, amelyek érvényesek az XHTML dokumentumokra.

Egy HTML oldalnak a MIME típusa `text/html`, míg egy XHTML oldalnak `application/xhtml+xml`. Miért említettem ezt meg? Azért, mert az Internet Explorer, a legelterjedtebb (vagy legalábbis az egyik legelterjedtebb) böngésző, ez utóbbit nem képes felismerni, magyarán nem képes megjeleníteni egy XHTML dokumentumot. Megtehetjük azt, hogy betartva a HTML-kompatibilitásra vonatkozó szabályokat küldjük el az XHTML dokumentumot, de ekkor elvesznek az XHTML előnyei. A HTML-kompatibilitásra vonatkozó elvekkel nem foglalkoznék, ezeket megtalálhatjuk az XHTML hivatalos referenciájának a „C” függelékében. Célszerű `application/xhtml+xml` módban küldeni azoknak a böngészőknek, amelyek ismerik ezt (ilyen például a Firefox), a többinek pedig `text/html`-ben. Egy kivétel: az XHTML 1.1-ben íródott oldalt nem lehet `text/html`-ben elküldeni. A `text/html`-ben elküldött XHTML oldal nem más, mint egy hibás HTML oldal,

mivel a `doctype`-pal, illetve az `xmlns`-szel és az `xml:lang`-gel a HTML nem tud mit kezdeni. Személy szerint annyit tudok mondani, hogy ha a böngésző támogatja az XHTML-t, akkor mindenképpen `application/xhtml+xml`-t használjunk, mert így élvezhetjük ki az XHTML nyújtotta előnyöket. Ilyenkor ugyanis nem a beépített HTML értelmező fut le, hanem az XML értelmező.
Példáimat én is XHTML oldalakként készítettem és mentettem el.

3.3. XUL

XML User Interface Language [10], [11]. A XUL-nak már az angol neve is egyértelművé teszi a funkcióját, különféle alkalmazások felhasználói felületének a létrehozására született a nyelv. A XUL az XML-en alapul, így természetesen annak minden jellemzője, sajátossága erre a nyelvre is érvényes és elérhető. Különféle hordozható felhasználói interfészek létrehozását teszi lehetővé. A XUL a Mozilla által kifejlesztett nyelv, amely létrehozásakor kifejezetten a Mozilla böngészőhöz jött létre azért, hogy hatékonyabb legyen a fejlesztése. Tudvalevő, hogy egy alkalmazást sokszor csak egy konkrét platformra fejlesztenek és optimalizálnak. Gyakran már ez is egy költséges dolog, nem is említve egy másik platformra történő átírás költségbeli és fejlesztésbeli vonzatait. Ezzel szemben viszont a XUL egy egyszerű, operációs rendszerektől független és keresztplatformos nyelv, aminek a használatához még csak programozási ismeret sem szükséges. Tehát egy interfésznek a XUL-lal történő implementálása és módosítása egyszerűen, könnyen és gyorsan történik. Mivel a Mozilla platform végzi egy XUL-beli alkalmazás renderelését, így minden olyan operációs rendszer alatt futni fognak ezek, amelyek alatt a Mozilla is fut. A XUL-ba más XML-alapú nyelvek kódjait is be lehet szúrni (például XHTML, SVG, stb.).
A XUL teljes implementációját jelenleg csak a Gecko tartalmazza, ami többek között a Firefoxban is működő böngészőmotor.
A XUL – mivel grafikus felhasználói felületek létrehozására szolgál – sokféle grafikus elem megvalósítását teszi számunkra lehetővé. Például: fák, szövegdobozok, checkboxok, hagyományos gombok, rádiógombok, címkék, menük, eszköztárak.

XUL alkalmazásokat többféle módon is létre lehet hozni, például:

- Firefox bővítmény: a bővítmények a böngészőhöz általában valamilyen plusz funkciót adnak. Ez megjelenhet eszköztárként, új menüpontként az Eszközök menüben, sőt akár az egér jobb gombjával történő kattintáskor felugró menüben is.
- XULRunner: ez gyakorlatilag egy különálló és önállóan végrehajtható fájl, egy futtatási környezet. Segítségével saját alkalmazást tudunk létrehozni, amely természetesen a XUL-ra és a Mozilla alkalmazási keretrendszerére épül. Minden XUL alapú alkalmazás képes futni rajta (például Firefox, Thunderbird).
- XUL csomag: könyvtáraknak és különböző fájloknak egy csoportja. A fájlok sokfélék lehetnek, például DTD, CSS, JavaScript, valamilyen képfájl. Egy átlagos csomag tartalmaz egy XUL fájlt, valamilyen scriptfájlt és egy CSS fájlt. Ezekből áll össze az alkalmazás, amely félig-meddig különálló, de a futtatáshoz szükség van feltelepített Mozilla böngészőre a gépen.
- távoli XUL alkalmazás: ebben az esetben a XUL kódot egy webszerverre helyezzük el, onnan tudjuk azt letölteni, illetve egy böngészőben megnyitni. Ez lehetővé teszi azt is, hogy webalkalmazásokhoz is használjuk a XUL-t.

Az első három esetben szükség van a felhasználó gépére való telepítésnek. Viszont ez így nem a legbiztonságosabb, mivel az ilyen alkalmazások a gépünkön hozzáférhetnek más fájlokhoz, esetlegesen fontos információkhoz, és távoli szervereket is el tudnak érni. Az utolsó esetben viszont léteznek megszorítások, és így sokminden le van korlátozva.

A bővítményeket egyetlen fájlként tudjuk letölteni, amelyek tartalmazzák a XUL fájlt, illetve a hozzá tartozó, az alkalmazás által használt szkripteket és képeket. A Mozilla alkalmazások (például az általam is használt Firefox), tartalmaznak bővítménykezelőt (vagy nevezhetjük kiterjesztéskezelőnek is), amelyekben az ilyen bővítményeket nagyon egyszerűen fel tudjuk telepíteni. Az XForms értelmező is így telepíthető fel.

A XUL dokumentumok általában .xul-ra végződnek, és ugyanúgy meg tudjuk nyitni őket Mozillában, mint más fájlokat. Egy XUL fájlt tehát betölthetünk a helyi gépen található fájlrendszerből, egy távoli weboldalról, a fentebb említett kiterjesztés segítségével, de önálló alkalmazásként is futtathatjuk (XULRunner).

3.4. MXML [5], [7]

Egy XML alapú leírónyelv, amely lehetőséget nyújt a fejlesztők számára, hogy Flex-alkalmazásokbeli összetevőket tervezzenek. Az MXML-t a Macromedia bocsájtotta ki 2004-ben [6].

Az MXML a Flex leírónyelve. Az alkalmazásfejlesztők az MXML-t úgynevezett gazdag internetes alkalmazások (RIA: Rich Internet Applications) fejlesztésére használják, általában az ActionScript nyelv lehetőségeivel kiegészítve. MXML-t használva meghatározhatjuk a különféle komponensek elhelyezkedését a képernyőn, például megadhatunk koordinátákat, méreteket.

Szerkeszteni akár egy egyszerű Notepadben is tudjuk, mivel ez egy szabvány leírónyelv, de persze az Adobe-nak a Flex Builderében is, amit erre a célra hoztak létre.

Az MXML segítségével az alkalmazásaink felhasználói felületét deklaratív módon tudjuk felépíteni. Ebből a szempontból hasonlít a HTML-hez, bár a szintaxisa egyértelműbb és részletesebb, és az elemkészlete is gazdagabb. Egy HTML oldalba a JavaScript segítségével tudunk interaktivitást bevinni, ennek a megfelelője az MXML esetében az ActionScript.

Lefordítva az MXML kódot, bináris flash fájlt, vagy másnéven SWF fájlt kapunk. A fordítást a szerver végzi akkor, amikor befut hozzá a kérés, tehát időben dinamikusan zajlik a fordítás. Ezután az SWF fájl elküldésre kerül a kliens gépre, és ott egy flash futtatására alkalmas lejátszó hajtja azt végre. Az előállítás a Flex Application Framework [7] (Flex Alkalmazási Keretrendszer) keretein belül történik. Ez tartalmazza az MXML és ActionScript kódokat, az előre megírt Flex osztálykönyvtárakat, és ezekből a Flex fordító előállítja a kimeneti kész fájlt.

Mint említettem, az MXML-lel együtt szokták használni az ActionScriptet is, így erről is szólnék pár mondatban.

ActionScript: ez egy objektum-orientált nyelv, amely elősegíti, kiegészíti a Flex alkalmazások fejlesztésének a menetét. Hasonlóan a JavaScripthez, ez is az ECMAScripten alapul. Használhatunk benne csomagokat, névtereket, osztályokat, ismeri az öröklődés fogalmát. Alkalmas az alkalmazások kliens-oldali logikájának a megírására. Egy ActionScript kód végrehajtása az AVM (ActionScript Virtual Machine) feladata, ami a Flash Playerbe van beépítve. Érdekes megjegyezni, hogy fordítás során az MXML kódból is ActionScript kód lesz.

3.5. XAML [8], [9]

Extensible Application Markup Language. Egy XML alapú nyelv, amely a Microsoft teméke. Ez a Microsoft következő generációs felhasználói felülete. Mivel Microsoft-termék, így nem platformfüggetlen.

Az XAML egy deklaratív nyelv, felhasználói felületek létrehozására szolgál, főképp a .NET Framework 3.0 keretrendszerben. Ennek a része a WPF (Windows Presetation Foundation), ami egy grafikus alrendszer és ami szintén az XAML-t használja az UI felépítéséhez. A Windows Vistában is az XAML szolgál elsődlegesen a grafikus felhasználói felületek létrehozásához.

XAML fájlokat olyan különféle fejlesztőeszközökkel hozhatunk létre és szerkeszthetünk, mint például a számunkra ismerős Microsoft Visual Studio. Ha nem rendelkezünk ilyen eszközzel a gépünkön, akkor akár egy sima szövegszerkesztővel is létrehozhatunk XAML dokumentumokat. Egy ilyen dokumentum először mindig egy köztes fájlba fordítódik (BAML, bináris XAML fájl), amelyet a .NET keretrendszer majd csak futási időben fordít és ellenőriz le (tehát időben dinamikus a folyamat). Egy XAML-ben írt dokumentumot mindig .xaml végű fájlba mentünk el.

Különválik a design, azaz a megjelenés, és a mögöttes logika. Ez a fejlesztés szempontjából nagyon hasznos, hiszen külön-külön lehet rajtuk dolgozni, és nincs szükség állandó egyeztetésre a két fejlesztői csapat között.

Az XAML-re épülve tetszőleges nyelven (például C#, Visual Basic) lehet írni WPF-es alkalmazásokat, webes alkalmazásokat.

Nem szükséges nagyon bonyolult feldolgozóprogram az XAML dokumentumokhoz. Ennek eredményeképpen sokféle olyan termék jelenik meg, amellyel XAML alapú alkalmazásokat tudunk létrehozni. Ezek közül sok a WPF területéhez kapcsolódik.

3.6. GTK+ [3],[4]

Ez nem egy konkrét leírónyelv, hanem egy nagyon jól használható, sokféle jellemzővel rendelkező eszközkészlet, amelynek segítségével grafikus felhasználói felületeket tudunk készíteni. Ezen felületek „cross-platformok”, magyarul hordozhatók a különféle platformok között (Windows, Mac OS, Unix, Linux).

A GIMP-hez (GNU Image Manipulation Program, egy grafikus szerkesztőprogram) lett kifejlesztve, a neve is ezt mutatja: GIMP ToolKit. C nyelven íródott a GNU LGPL licensze alatt, amely keretein belül szabadon fejleszthetünk programokat. A GTK+ négy könyvtáron alapszik: GLib, Pango, Cairo, ATK.
Sokféle nyelvhez lehet felhasználni (Perl, Python, C, C++, C#, Java, stb.). Nagyon nagy fejlesztői közösséggel rendelkezik, a kódhoz mindenki hozzátehet.
Manapság már nagyon sok szoftver fejlesztésében használják, például a GNOME-projektben is. A GNOME (GNU Network Object Model Enviroment) egy grafikus felhasználói felület, amellyel például a Linuxban találkozhatunk.
A GTK+-nak a legfrissebb változata a 2.12.3. Ez már a GTK+ 2, amiről érdemes tudni, hogy nem kompatibilis a GTK+-al.
Sokféle elemet lehet benne használni, például menük, ablakok, gombok, eszköztárak, stb.
A felületek megtervezése a programkódon belül eléggé hely- és időigényes feladat, ezért célszerű mondjuk a Glade-t használni, aminek segítségével a felületet grafikusan meg tudjuk tervezni. A Glade az elkészült felületből forráskódot tud generálni. Létezik neki egy Libglade nevű programkönyvtára, amelynek segítségével az alkalmazás futásakor képes felépíteni annak felületét, méghozzá a Glade által generált XML fájl alapján. A Libglade segítségével készített felületek egyrészt az egyes vezérlőelemek elnevezésén keresztül, másrészt az eseménykezelő függvényeken keresztül kapcsolódnak az alkalmazás kódjához. Így tudunk hivatkozni a felület elemeire, illetve beállított eseményeire.

4. XForms [18], [19], [20], [24]

Az XForms a HTML űrlapok következő generációja. Attól eltérően az űrlapok megvalósítását XML alapokon oldja meg, és annál jóval sokoldalúbban és rugalmasabban használható. Az XForms legelső ajánlása 2003-ban jelent meg, legutóbbi kiadásai: az XForms 1.0-nak a harmadik kiadása, illetve az XForms 1.1, mindkettő 2007 végén jelent meg. Az 1.1 sok újítást tartalmaz a korábbi változathoz képest. Itt most az 1.0-s verzióról lesz szó, mivel én is azt vettem alapul.

Az XForms főbb jellemzői:

- a HTML-beli űrlapok egyfajta leváltásának is tekinthetjük: az űrlapok elsősorban arra szolgálnak, hogy webes alkalmazásba beépítve őket adatokat kérjünk be a felhasználóktól. Erre szolgálnak a HTML űrlapok is, de az XForms ennél egy eszközgazdagabb és jobban használható nyelv
- hivatalos W3C ajánlás
- az űrlap megjelenítését és az adatait elkülöníti egymástól: gyakorlatilag az adatok létrehozása XML alapokon történik, míg a megjelenítéshez HTML-t vagy XHMTL-t használ
- XML formátumban definiálja az adatokat, és XML dokumentumban tárolja és továbbítja őket más alkalmazások vagy felhasználók számára
- Unicode alapú
- mivel köthető az XML-hez, így használhatunk benne olyan szabványokat, amelyek kapcsolódnak az XML-hez, például: XPath, XML Schema
- platformfüggetlen
- eszközfüggetlen: mivel az adat- és a megjelenítési rész elkülönül egymástól, így az adatmodellt mindenféle eszközön használhatjuk, a megjelenítést pedig az adott eszközhöz, illetve annak felhasználói interfészéhez kell igazítani, használhatjuk mobiltelefonokon, PDA-n, stb.
- más XML-es alkalmazásokba is közvetlenül elhelyezhetünk XForms elemeket
- az XForms kód önmagában nem működőképes, más nyelvekbe kell integrálni

Ezek az XForms főbb jellemzői. Most menjünk bele egy kicsit részletesebben is a bemutatásába.

Mint említettem, külön van választva az adatok definiálása és a megjelenítése:

- **XForms model**: itt az adatok leírása történik, megadjuk az adatok tartalmát, egyéb jellemzőket, ennek a felépítése egy XML dokumentum felépítéséhez hasonlítható
- **Xforms user interface**: ez a rész felelős az adatok megjelenítéséért, és az adatok beviteléért, itt jelennek meg a felület különböző elemei

Nézzünk egy általános `model` részt:

```
<model>
  <instance>
    <ember>
```

```
        <name/>
      </ember>
    </instance>
    <submission id="urlapkuld" action="kuld.asp" method="get" />
  </model>
```

Az `instance` részek között adjuk meg az adatokat, amelyeket az űrlapról gyűjtünk be, vagy ott felhasználunk, mivel lehetőség van az itteni mezőket nem üresen hagyni, én is így adtam meg egyes kezdőértékeket a programjaimban. Ez a rész gyakorlatilag egy XML dokumentumot definiál. Ha a fenti kódrészlethez társítunk egy beviteli mezőt és ott bekérjük a nevet, akkor már így fog kinézni: `<name>Nagy Sándor</name>`.

A példában látható még egy `submit` rész, ennek segítségével mondjuk meg, hogy mit csináljunk akkor, ha elküldjük az adatokat. Ehhez a részhez a megjelenítési részben egy submit-gombot szokás társítani, majd mindjárt látjuk, hogy hogyan. Ebben a részben meg van adva egy `id`, ami egy egyszerű azonosító, egy URL, ez hajtódik végre az elküldéskor, ebben megírhatjuk, hogy ilyenkor mi is történjen. Van még egy `method`, ami megmondja, hogy milyen metódus lesz használva az elküldéskor (például POST, GET), a *get* az elküldött adatokat az URL-ben helyezi el egy kérdőjellel elválasztva. Ebben a részben még más jellemzőket is megadhatunk, például minden adathoz megadhatunk valamilyen XML Schema általi típust a `bind` elem segítségével.

Nézzünk a fenti `model` részhez kapcsolódó `interface` részt:

```
  <input ref="name"><label>Adja meg a nevét:</label></input>
  <submit submission="urlapkuld"><label>Küld</label></submit>
```

Megadtunk egy kis beviteli mezőt a névnek, ahol a névre egyszerűen `name`-mel hivatkozunk, így az itt megadott név a beolvasás után abba az adatrészbe fog kerülni. A küldésnél is hasonló a helyzet, csak itt nem `ref`, hanem `submission` után írjuk be az azonosítót. Ez egy nyomógombot fog megjeleníteni, amire rákattintva a fentebb definiált ASP kód fog végrehajtódni a megadott metódus szerint.

XHTML-be mentve a kódot, a `model` részt mindig a `<head>`-ben, a megjelenítési részt pedig mindig a `<body>`-n belül kell megadni.

A fenti kódrészlet egy egyszerű példának is felfogható, természetesen, ha beágyazzuk valamilyen nyelven megírt dokumentumba (például XHTML). Ahhoz, hogy egyáltalán megjelenítődjön és működjön, XHTML vagy XML formátumban kell elmentenünk az XForms-ot tartalmazó kódot, mivel ilyenkor automatikusan `application/xhtml+xml` vagy

`text/xml` lesz a MIME-típus, ami szükséges ahhoz, hogy helyesen értelmezze a kódunkat a feldolgozó. Ezen kívül szükséges egy kiegészítés is a böngészőbe, aminek segítségével az XForms-os részek is értelmezésre és megjelenítésre kerülnek. Sajnos még nem tartunk ott, hogy a böngészőkben már alapból implementálva lenne. A Firefoxhoz egyszerűen a Mozilla oldaláról le tudjuk tölteni ezt a kis kiegészítést, amit azután fel kell telepíteni az Eszközök menü Kiegészítők fülére kattintva. Az Internet Explorerhez is van lehetőség kiegészítő letöltésére, én egy `formfaces.js` nevű JavaScript fájlt találtam, amelyet úgy lehet használni, hogy a `<head>` elemen belül be kell linkelnünk, mint egy külső JavaScript-fájlt:

```
<script type="text/javascript" src="fomfaces.js"></script>
```

Nagyon fontos megemlíteni, hogy az XForms elemeit a kódban csak úgy használhatjuk, ha megadjuk az elemek neve előtt az XForms-os névtér nevét. A névteret még a kód elején, a `<html>` elemen belül kell megadni, ami nagyon egyszerűen történik, az alábbiak szerint:

```
<html … xmlns:xf="http://www.w3.org/2002/xforms" …>
```

Ezek után lássuk, hogy a fenti példa hogyan is néz ki:

2. ábra

A megjelenítéshez Mozilla Firefoxot használtam.

Az XForms-ban kihasználjuk az **XPath** [17] lehetőségeit is. Mint a fenti példában is láttuk, az `instance` részben létrehozott adatra, az embernek a nevére egyszerűen a nevével (`name`) hivatkoztam. Az XPath biztosít lehetőséget számunkra, hogy az adatlétrehozó és a megjelenítési rész között kapcsolatot teremtsünk, mivel az XPath nem más, mint egy olyan szintaktika, amivel egy XML dokumentum elemeire tudunk hivatkozni. Szintén a W3C szabványa. A DOM-hoz hasonló logikai modellt definiál egy XML dokumentumhoz, amely egy fa modellt jelent. Ennek a fa modellnek a csomópontjait tudjuk azonosítani, elérni. Az XPath-el történő összekapcsolást bindingnek nevezik, amit ha magyarra akarnánk fordítani, akkor szintén összekapcsolást, összekötést jelentene. Lássunk egy példát:

```
<model>
  <instance>
    <auto>
      <marka>
        <tipus/>
```

```
        </marka>
      </auto>
    </instance>
  </model>
  …
  <input ref="marka/tipus"><label>Típus</label></input>
```

Látható, hogy itt a `tipus` elemre a `marka/tipus` útvonallal hivatkozunk, és hogy ezt az adott elemen belül a `ref` paraméter segítségével tudjuk megtenni. Abszolút utat megadva úgy is írhattuk volna, hogy:

```
<input ref="/auto/marka/tipus"><label>Típus</label></input>
```

Van egy olyan lehetőségünk is, hogy egy adott adatelemhez egy azonosítót rendelek hozzá, így nem kell mindig hosszasan begépelni az XPath útvonalát. Ezt a `model` részen belül, az `instance` rész után tudom megtenni, az alábbiak szerint:

```
<bind nodeset="/auto/marka/tipus" id="autotipus"/>
```

Ezek után a megjelenítési részben már nem `ref`-el hivatkozok rá, hanem `bind`-el, így:

```
<input bind="autotipus"><label>Típus</label></input>
```

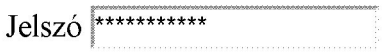
Nyilván összetettebb alkalmazásoknál, ahol sok az elem az `instance`-on belül, ott már érdemes ezt a módszert használni, mert könnyebb kezelhetőséget tesz lehetővé.
Az XPath-tal nem csak útvonalakat adhatunk meg, hanem például megszorításokat, számításokat is.

Most röviden tekintsük át, hogy az XForms felhasználói felületén (az UI részen) belül milyen beviteli elemeket tudunk használni. Ezek az elemek böngészőfüggően jelennek meg, tehát a böngészőre van bízva, hogy hogyan jelenik meg, az XForms-ban ezt nem lehet külön megadni, egy platformfüggetlen nyelvnél amúgy sem lehet egységes megjelenést elérni.
Vegyük sorra először az úgynevezett form control elemeket (magyarul űrlapvezérlőknek lehetne őket fordítani):

- **Input:** ez a legegyszerűbb és leggyakrabban használt, egy egysoros szöveges beviteli mezőt hoz létre, feljebb erre már láthattunk példát
- **Submit:** ez a másik leggyakrabban használt elem, mert ez a gomb szolgál a küldésre, a fenti példában ez is szerepelt, így ennek sem térek ki a szintaktikájára és a megjelenésére
- **Label:** ez egy olyan elem, amelyet az összes többi elembe be lehet építeni. Véleményem szerint célszerű is megadni, hiszen jó, ha tudjuk, hogy például egy

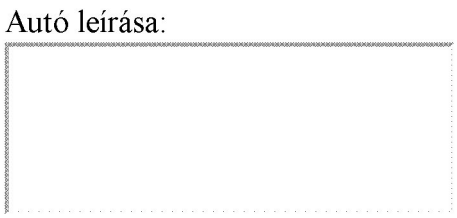
`input` mezőbe milyen adatot kell beírnunk. Ez az elem gyakorlatilag felcímkézi azt az elemet, amelybe beépítjük, méghozzá egy leíró címkét ad hozzá, amiből tudjuk, hogy az adott űrlapelem mire szolgál, vagy például az űrlap egyes elemeit is meg tudjuk adni a segítségével (például `select`-nél).

- **Secret:** hasonlatos az `input`-hoz, annyi a különbség, hogy ez jelszavak bekérésére szolgál, és itt a beírt szöveg karakterei nem jelennek meg, helyettük csak csillagokat látunk. A szintaxisát egy nagyon egyszerű példán keresztül nézhetjük meg:
```
<secret bind="pw"><label>Jelszó</label></secret>
```



3. ábra

- **Textarea:** ezáltal egy olyan szövegbeviteli mezőt hozunk létre, amelyben többsoros szöveg bevitelére van lehetőségünk, amely, ha szükséges, vízszintes és függőleges irányban is továbbgördíthető. A szövegbeviteli mezőre egy képes példa:
```
<textarea ref="auto/leiras"><label>Autó leírása:</label></textarea>
```



4. ábra

- **Trigger:** a trigger egy olyan gomb, amelyre rákattintva valamilyen művelet hajtódik végre. Például: rákattintunk a gombra, és ennek hatására lefut egy JavaScript-kód, vagy egy adatelem értéke átállítódik. Nézzünk a triggerre egy példát:
```
<trigger ref="kiir"><label>Írd ki!</label></trigger>
```

5. ábra

- **Output:** XForms-beli adatok kiíratására, megjelenítésére szolgáló elem. Például: `<b><output ref="name" /></b>`. Itt megjelenik a név, amit tudunk formázni is, esetünkben kövér betűkkel lesz kiírva.

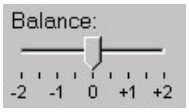
John Newman

6. ábra

- **Upload:** mint a neve is mutatja, ez az elem fájlok feltöltésére szolgál. Például:
  ```
  <upload ref="auto/kep" mediatype="image/*"><label>Kép kiválasztása:
  </label></upload>
  ```
  A képen látszik, hogy lehetőségünk van beírni a kiválasztott képet, böngészni a helyi fájlrendszerben, vagy a már beírt nevet törölni.

7. ábra

- **Range:** ezzel egy amolyan „csúszkát” hozhatunk létre, amelyen kiválaszthatunk egy értéket egy megadott intervallumon belül. Nézzünk erre is egy rövid példát:
  ```
  <range ref="ertekbeallit" start="-2.0" end="2.0" step="0.5"><label>
  Balance:</label></range>
  ```
  Ez a példa az XForms referenciából lett kiragadva, mivel az általam írtat elmentve maga a lényeg, a „csúszka”, sehogyan sem akart megjelenni a képen (csak MDI-formátumban, de azzal nem sokra megyek), Firefoxban egyébként más a megjelenítése.

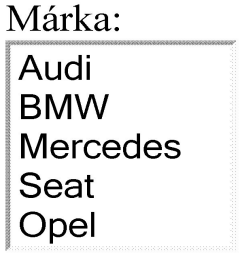


8. ábra

- **Select1:** egy listából egy elemet tudunk kiválasztani a segítségével. Például:
  ```
  <select1 ref="auto/marka"><label>Márka:</label>
  <item><label>Audi</label><value>audi</value></item>
  <item><label>BMW</label><value>bmw</value></item>
  <item><label>Mercedes</label><value>merci</value></item></select1>
  ```
  Mint látható, az egyes választási lehetőségeket az `item`-ek között adhatjuk meg, azon belül megadunk egy címkét és egy értéket, amit majd a `marka` tartalmazni fog.

9. ábra

- **Select:** nagyon hasonló az előzőhöz, mindössze abban különbözik, hogy itt egyszerre akár több elem kiválasztására is van lehetőség. A szintaktikája abban különbözik, hogy `select1` helyett `select` van.



10. ábra

Utóbbi kettőnél lehetőségünk van arra, hogy a megjelenítésen változtassunk. Ezt egy `appearance` attribútum segítségével tudjuk megadni, amelynek értéke lehet `full`, `minimal` vagy `compact`, mindegyiknél más-más a kinézet és természetesen a méret is.

Ezek az űrlapok fő elemei, ezekből már össze tudunk rakni egy űrlapot. Ezek azok, amelyek ténylegesen megjelennek, de ezeken kívül vannak még továbbiak is. A fenti esetekben is láthattunk már erre példát, elég csak a `value`, az `item` vagy a `label` elemekre gondolnunk. De léteznek mások is, például: `hint`, `alert`, `help`, `filename`, stb.

Az XForms támogatja az XML Schema-beli adattípusokat, 1-2 kivétellel. Ahhoz, hogy ezeket használni is tudjuk, meg kell adni az XML Schema névterét is az alábbiak szerint:

```
<html xmlns:xf="http://www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

Ilyenkor megadhatunk az adatelemeknek típust is:

```
<xf:instance>
  <auto>
    <marka xsi:type="xsd:string"/>
    <ar xsi:type="xsd:integer"/>
</xf:instance>
```

Megadhatjuk a típust másképpen is, az `instance` rész után is:

```
<bind nodeset="/auto/marka" type="xsd:string"/>
<bind nodeset="/auto/ar" type="xsd:integer"/>
```

Van négy adattípus, amely nem származtatott, hanem saját (XForms adattípusok):

- `listItem`
- `listItems`
- `dayTimeDuration`

- `yearMonthDuration`

Az adatelemekhez megadhatunk különböző tulajdonságokat, mint például:

- **type:** segítségével típusmegszorítást adhatunk meg, fentebb már láthattunk erre vonatkozó példákat. Ha nincs megadva, akkor az alapértelmezett az `xsd:string`.
- **required:** ennek alapértéke `false`, de ha `true`-ra állítjuk be, akkor azt jelenti, hogy az adott adatot nem hagyhatjuk üresen. Például egy űrlapon beállíthatjuk, hogy a nevet bekérő inputmezőt kötelező kitölteni.
- **readonly:** a neve magáért beszél, az adott adat csak olvasható, nem lehet az értékét módosítani. Alapértelmezés szerint `false`.
- **constraint:** megszorításokat adhatunk meg az adatra vonatkozóan, és ha ezeket nem teljesíti, akkor nem lesz érvényes az érték
- **calculate:** valamilyen érvényes XPath kifejezést adhatunk meg, ami kiszámol egy értéket az adatelemhez

Az XForms-ban különféle események léteznek, amelyek például elősegítik az állapotok megváltoztatását. Ezeknek sok fajtája van, így nem is mennék bele a jellemzésükbe. Helyette csak egy-két példát említenék meg:

- **DOMActivate:** ez az esemény akkor jön létre, ha mondjuk entert ütünk valahol, vagy megnyomunk egy gombot
- **xforms-value-changed:** akkor keletkezik, ha egy adatelem értékét megváltoztatjuk
- **DOMFocusIn:** az adott űrlapelem fölé megyünk az egérrel
- **xforms-select:** ha egy `select`, `select1` vagy `switch`-beli elem kiválasztódik
- stb., sok más esemény van még deklarálva az XForms-ban, érdemes áttekinteni a referenciát, ha kíváncsiak vagyunk rájuk

Az események alapjául szolgálnak néhány műveletnek, ezek közül néhány:

- **message:** egy üzenetet jelenít meg a felhasználó felé, például:

```
<input ref="marka"><label>Írja be a márkát: </label><message
level="ephemeral" event="DOMFocusIn">Írja be az autó
márkáját!</message></input>
```

Ha a beviteli mező fölé megyek, akkor amolyan „tool tip”-ként jelenik meg a beállított üzenet. A `level`-ben tudjuk megadni, hogy hogyan jelenjen meg az üzenet.

- **setvalue:** értéket tudunk beállítani a segítségével, például:

```
<input ref="labmeret"><label>Méret</label><setvalue value="42"
event="xforms-ready"></input>
```

  Beállítja 42-re az értéket, mikor az űrlap megnyílik.
- **action:** ezen belül megadhatunk egyéb műveleteket, amelyek egymás után hajtódnak végre, például:

```
<action event="DOMActivate"><reset model="automodel"/><setvalue
ref="auto/marka" value="Renault"/></action>
```

Különféle függvények használatára is van lehetőségünk. Ezek az XPath programkönyvtár függvényei. Például:

- **min(node-set):** az elemek minimumát adja vissza
- **max(node-set):** az elemek maximumát adja vissza
- **avg(node-set):** az elemek átlagát adja meg
- **if(booleantest,string1,string2):** ha a `booleantest` igaz, akkor a `string1`-gyel, ha hamis, akkor a `string2`-vel tér vissza

Léteznek még olyan eszközök az XForms-ban, amelyeket az űrlapvezérlő-elemekkel (`input`, `trigger`, stb) kombináltan tudunk felhasználni. Néhány példa:

- **group:** segítségével csoportokat tudunk létrehozni, ezáltal összetetteb hierarchia felépítésére nyílik módunk
- **switch:** ismerős lehet más programnyelvekből, `case` elemeket tartalmaz, amelyek közül egy adott időben csak egy hajtódhat végre
- **case:** elhelyezhető `group`-ban, `switch`-ben, vagy űrlapvezérlő elemekben is. Valamilyen feltételes megjelenítést jelöl, például megadjuk egy `trigger`-en belül, és amikor az végrehajtódik, akkor a `case`-ben megadott `switch`-beli ágra ugrik, és végrehajtja az ottani elemeket.
- **toggle:** segítségével kiválaszthatunk egy `case`-t a `switch`-ben szereplő alternatívák közül

Lássunk egy konkrét példát is az itt elmondottakra:

```
<switch>
  <case id="marka" selected="true">
    <input>
      <label>Adja meg az autó márkáját: </label>
      <toggle ev:event="DOMActivate" case="kiir" />
    </input>
    </case>
    <case id="kiir" selected="false">
      <output ref="auto/marka" />
      <trigger>
        <label>Módosít</label>
        <toggle ev:event="DOMActivate" case="marka" />
      </trigger>
    </case>
</switch>
```

Ez a kódrészlet azt csinálja, hogy beolvassa az autó márkáját, majd kiírja azt, de közben lehetőséget is ad a módosításra egy `trigger` segítségével. A beolvasás történik először (a `selected="true"` is ezért kell), majd pedig innen ugrunk a másik ágra, ahonnét szintén van lehetőség a visszaugrásra.

Röviden ennyit az XForms-ról. Ez a bemutatás természetesen nem törekedett a teljességre, célja inkább az volt, hogy bemutassa a nyelv lehetőségeit, főbb eszközeit, amelyek alapján már elindulhatunk az XForms-os űrlapkészítés útján. Véleményem szerint nem egy bonyolult eszközről van szó, az alapokat könnyű elsajátítani. Aki teljes terjedelmében meg akarja ismerni a nyelvet, annak javaslom a hivatalos referencia átolvasását.

5. Példák felhasználói felületekre

Az alábbiakban három felhasználói felületet fogunk látni, amelyek az XForms, illetve az XHTML alapjain lettek megvalósítva. Mindhárom egy-egy egyszerűbb játék, viszonylag egyszerű grafikus felülettel. A weblapok Firefox böngészőre lettek optimalizálva.

5.1. Gyümölcscsereberélős játék

- **játék lényege:** adott három különböző fajta gyümölcs, amelyeket egymással lehet cserélgetni, méghozzá úgy, hogy ha az egyik gyümölcsből szeretnénk kapni, akkor kapunk belőle kettőt, míg a másik kettőből pedig elveszünk egyet-egyet
- **játék célja:** ügyesen cserélgetve a gyümölcsöket, érjük el, hogy csak egyetlen fajta gyümölcsünk maradjon

11. ábra

Íme itt látható a játék képe, az aktuális darabszámokkal, illetve a nyomógombokkal, amelyekkel az adott gyümölcsből lehet kérni. A játék végén két gyümölcsnél is 0-s darabszámot kell látnunk.

A forráskód:

Itt a weblap elején szokásos megadások következnek:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xf="http://www.w3.org/2002/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xml:lang="en">
```

A <head> rész:

```
<head>
<xf:model id="modell">
  <xf:instance xmlns="" id="adatok">
   <gyumolcs>
    <alma>13</alma>
```

```
      <korte>46</korte>
      <barack>59</barack>
    </gyumolcs>
  </xf:instance>
  <xf:bind nodeset="/gyumolcs/alma" type="xsd:integer"/>
  <xf:bind nodeset="/gyumolcs/korte" type="xsd:integer"/>
  <xf:bind nodeset="/gyumolcs/barack" type="xsd:integer"/>
  </xf:model>
  <script type="text/javascript" src="gyumolcs.js"/>
  </head>
```

A fejben kell deklarálnunk az XForms-os `model` részt. Az `instance`-nál sokszor meg szoktuk adni a névtelen névteret, ez az alapértelmezett névteret jelenti. Aztán megadjuk a háromféle gyümölcsöt, az elemek a darabszámot reprezentálják. Végül pedig megadjuk, hogy ez az érték `integer` típusú lehet (ez egy XML Schema-beli típus), illetve egy külső JavaScript fájlt linkelünk be, amely a működéshez elengedhetetlen függvényeket tartalmaz. Lehetett volna a dokumentumon belül is létrehozni, de így átláthatóbb.

A `<body>` rész:

```
  <body>
    <b><font size="6">Gyümölcscsereberélős játék</font></b>
    <br />
    <br />
    <table border="5">
      <tr>
        <td align="center">Alma</td>
        <td align="center">Körte</td>
        <td align="center">Barack</td>
      </tr>
      <tr>
        <td align="center"><xf:output ref="/gyumolcs/alma"/></td>
        <td align="center"><xf:output ref="/gyumolcs/korte"/></td>
        <td align="center"><xf:output ref="/gyumolcs/barack"/></td>
      </tr>
      <tr>
        <td>
          <xf:trigger id="alma">
            <xf:label>Kérek!</xf:label>
```

```
            <xf:load resource="javascript:alma_kap()"
            ev:event="DOMActivate"/>
          </xf:trigger>
        </td>
        <td>
          <xf:trigger id="korte">
            <xf:label>Kérek!</xf:label>
            <xf:load resource="javascript:korte_kap()"
            ev:event="DOMActivate"/>
          </xf:trigger>
        </td>
        <td>
          <xf:trigger id="barack">
            <xf:label>Kérek!</xf:label>
            <xf:load resource="javascript:barack_kap()"
            ev:event="DOMActivate"/>
          </xf:trigger>
        </td>
      </tr>
    </table>
    <br />
    <b>A játék lényege:</b> Ha kérsz egy gyümölcsből,akkor kapsz belőle
  kettőt,a másik kettőből pedig elvesz egyet-egyet. Természetesen ez csak
  akkor működik,ha mind a kettőből lehet még elvenni.
    <br />
    <b>A játék célja:</b> Érd el,hogy csak egyféle gyümölcsöd maradjon.
  </body>
  </html>
```

Megadunk egy táblázatot, az első sorában kiírjuk a gyümölcsök neveit, a másodikban a darabszámokat az `output` segítségével, ez mindig frissül, ha kérek valamelyik gyümölcsből. Az alsó sorban pedig nyomógombokat helyezek el, amelyekre rákattintva a megfelelő JavaScript függvény töltődik be, és az adott gyümölcsből ennek hatására kapunk kettőt, a másik kettőből pedig elveszünk egyet-egyet.

És most lássuk a Javascript fájlt:

```
function alma_kap(){
```

Ebbe a változóba fog kerülni az XForms `model` része, ami a forrásdokumentum `head` részében van létrehozva:

```
var model=document.getElementById("modell");
```

Az `adatok` nevű `instance` rész kerül bele:

```
var ins=model.getInstanceDocument("adatok");
```

A „*” segítségével az összes `ins` változóbeli elem belekerül ebbe a változóba:

```
var gy=ins.getElementsByTagName("*");
```

A `gy` változó elemeit sorra vesszük, és az értékeit eltároljuk ezekben a darabszámot reprezentáló változókban. A nulladik elem a `gyumolcs` nevű dokumentumelem:

```
var alma_db=parseInt(gy[1].firstChild.nodeValue);
var korte_db=parseInt(gy[2].firstChild.nodeValue);
var barack_db=parseInt(gy[3].firstChild.nodeValue);
```

Megvizsgáljuk, hogy lehetséges-e még almát kapnunk. Ennek az a feltétele, hogy a másik két gyümölcsből még legyen legalább egy-egy darab. Ha a feltétel igaz, megtörténik a csere, egyébként vége a játéknak:

```
if(korte_db>=1 && barack_db>=1) {
  alma_db+=2;
  korte_db-=1;
  barack_db-=1;
}
else{
  alert("Vége a játéknak!");
  return;
}
```

Az aktuális darabszámokkal felülírom az eredeti darabszámokat. Az eredeti értékeket csak így tudom elérni, mivel az `alma_db`, a `korte_db` és a `barack_db` pusztán helyi változók:

```
gy[1].firstChild.nodeValue=alma_db;
gy[2].firstChild.nodeValue=korte_db;
gy[3].firstChild.nodeValue=barack_db;
```

Ezen függvények segítségével frissítjük az egész XForms-modellünket, az adatok nem csak az `instance` részen belül kerülnek módosításra, hanem az űrlapvezérlő elemeknél is (például az `output`-nál):

```
model.rebuild();
model.recalculate();
```

```
    model.refresh();
```

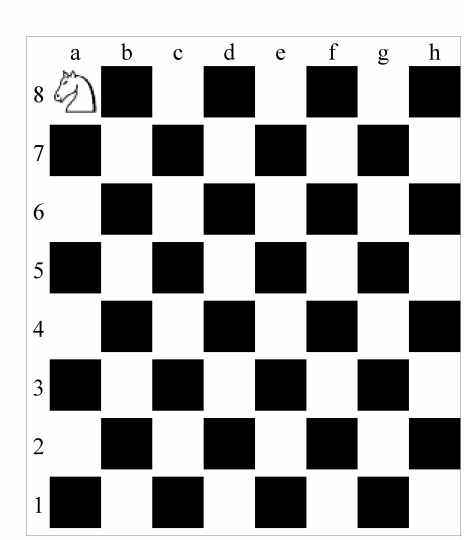
Ha a másik két gyümölcs elfogyott, akkor megnyertük a játékot:

```
    if(korte_db==0 && barack_db==0)
      alert("Gratulálok! Csak egyféle gyümölcs maradt,így nyertél!");
      return;
  }
```

A körtéhez és a barackhoz tartozó függvény szintén hasonlóan épül fel, így azokat nem írnám le.

5.2. Huszáros játék

- **játék lényege:** van egy huszárfiguránk egy sakktáblán, és azzal tudunk szabályos lépések megtételével lépdelni a táblán
- **játék célja:** a kiindulási pozíció a bal felső mező, célunk, hogy eljussunk a jobb alsó mezőbe



12. ábra

A képen látható a sakktábla, amely gyakorlatilag egy HTML-beli `table`, amelynek a mezőiben egy-egy `trigger` van megvalósítva.

Lássuk a forráskódot:

Szokásos deklarációk:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xf="http://www.w3.org/2002/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xml:lang="en">
```

Az adatelemek létrehozása, sorrendben az aktuális pozíció vízszintes és függőleges koordinátái, aztán annak a mezőnek a koordinátái, ahová lépni akarunk (és amelyikre egyúttal kattintottunk is), az aktuális mező színe, és annak a mezőnek a színe, ahová lépni szeretnénk:

```
<head>
<xf:model id="modell">
  <xf:instance xmlns="" id="adatok">
   <koordinata>
     <vizakt>1</vizakt>
     <fuggakt>1</fuggakt>
     <vizhova/>
     <fugghova/>
     <szinakt>0</szinakt> <!--az egyszerűség kedvéért legyen a "0" a
     fehér, az "1" a fekete -->
     <szinhova/>
   </koordinata>
</xf:instance>
```

Típusok megadása:

```
<xf:bind nodeset="/koordinata/vizakt" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/fuggakt" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/vizhova" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/fugghova" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/szinakt" type="xsd:binary"/>
<xf:bind nodeset="/koordinata/szinhova" type="xsd:binary"/>
</xf:model>
```

Stíluslap-információk megadása:

```
<style type="text/css">
  #fekete {background-color: black}
  #elso {text-align: center}
  #cim {text-align: center}
</style>
```

Külső JavaScript fájl belinkelése:

```
<script type="text/javascript" src="huszar.js"/>
```

```
</head>
```

A <body> rész:

```
<body>
  <h1 id="cim">"Huszáros" játék</h1>
  <table border="1" cellpadding="0" cellspacing="0" align="center"
  rules="none">
    <tr>
      <td />
```

A sakktábla első sora, ami az oszlopjelöléseket tartalmazza (angol „abc” kisbetűi):

```
      <td align="center">a</td>
      <td align="center">b</td>
      <td align="center">c</td>
      <td align="center">d</td>
      <td align="center">e</td>
      <td align="center">f</td>
      <td align="center">g</td>
      <td align="center">h</td>
    </tr>
    <tr>
```

A táblázat második sorának első eleme, ami a sakktábla legfelső sorát jelöli, az első oszlop ezeket a számokat tartalmazza, (többi sor elején hasonlóan):

```
      <td id="elso" width="5%">8</td>
```

Második sor második eleme, ami a sakktábla első mezője. Egy `trigger` található benne, aminek a megjelenését a lehető legkisebbre állítottam az `appearance` attribútummal, és egy képet „húztam rá” a gombra. Amikor rákattintok a gombra, akkor az `action`-ön belüli műveletek hajtódnak végre, beállítódik a célmező koordinátája és színe, majd pedig lefut a `lep` nevű JavaScript függvény:

```
      <td>
        <xf:trigger appearance="minimal">
          <img src="feherh.bmp" />
          <xf:action ev:event="DOMActivate">
            <xf:setvalue ref="/koordinata/szinhova" value="0"/>
            <xf:setvalue ref="/koordinata/vizhova" value="1"/>
            <xf:setvalue ref="/koordinata/fugghova" value="1"/>
            <xf:load resource="javascript:lep()"/>
          </xf:action>
```

```
      </xf:trigger>
    </td>
```

Erre az azonosítóra azért van szükség, hogy a fekete mezők cellái is feketék legyenek, mivel bárhogyan alakítottam, alul mindig maradt egy kis fehér csík:

```
<td id="fekete">
```

A sor többi része:

```
     <xf:trigger appearance="minimal">
       <img src="fekete.bmp" />
       <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="/koordinata/szinhova" value="1"/>
         <xf:setvalue ref="/koordinata/vizhova" value="1"/>
         <xf:setvalue ref="/koordinata/fugghova" value="2"/>
         <xf:load resource="javascript:lep()"/>
       </xf:action>
     </xf:trigger>
   </td>
   <td>
     <xf:trigger appearance="minimal">
       <img src="feher.bmp" />
       <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="/koordinata/szinhova" value="0"/>
         <xf:setvalue ref="/koordinata/vizhova" value="1"/>
         <xf:setvalue ref="/koordinata/fugghova" value="3"/>
         <xf:load resource="javascript:lep()"/>
       </xf:action>
     </xf:trigger>
   </td>
   <td id="fekete">
     <xf:trigger appearance="minimal">
       <img src="fekete.bmp" />
       <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="/koordinata/szinhova" value="1"/>
         <xf:setvalue ref="/koordinata/vizhova" value="1"/>
         <xf:setvalue ref="/koordinata/fugghova" value="4"/>
         <xf:load resource="javascript:lep()"/>
       </xf:action>
     </xf:trigger>
   </td>
   <td>
```

```
  <xf:trigger appearance="minimal">
    <img src="feher.bmp" />
    <xf:action ev:event="DOMActivate">
      <xf:setvalue ref="/koordinata/szinhova" value="0"/>
      <xf:setvalue ref="/koordinata/vizhova" value="1"/>
      <xf:setvalue ref="/koordinata/fugghova" value="5"/>
      <xf:load resource="javascript:lep()"/>
    </xf:action>
  </xf:trigger>
</td>
<td id="fekete">
  <xf:trigger appearance="minimal">
    <img src="fekete.bmp" />
    <xf:action ev:event="DOMActivate">
      <xf:setvalue ref="/koordinata/szinhova" value="1"/>
      <xf:setvalue ref="/koordinata/vizhova" value="1"/>
      <xf:setvalue ref="/koordinata/fugghova" value="6"/>
      <xf:load resource="javascript:lep()"/>
    </xf:action>
  </xf:trigger>
</td>
<td>
  <xf:trigger appearance="minimal">
    <img src="feher.bmp" />
    <xf:action ev:event="DOMActivate">
      <xf:setvalue ref="/koordinata/szinhova" value="0"/>
      <xf:setvalue ref="/koordinata/vizhova" value="1"/>
      <xf:setvalue ref="/koordinata/fugghova" value="7"/>
      <xf:load resource="javascript:lep()"/>
    </xf:action>
  </xf:trigger>
</td>
<td id="fekete">
  <xf:trigger appearance="minimal">
    <img src="fekete.bmp" />
    <xf:action ev:event="DOMActivate">
      <xf:setvalue ref="/koordinata/szinhova" value="1"/>
      <xf:setvalue ref="/koordinata/vizhova" value="1"/>
      <xf:setvalue ref="/koordinata/fugghova" value="8"/>
```

```
            <xf:load resource="javascript:lep()"/>
          </xf:action>
        </xf:trigger>
      </td>
    </tr>
…
```

A többi sor megvalósítása is ugyanígy készült, így azokat nem írnám le.

```
</table>
  <br />
  <b>Játék célja: </b>A huszár figurával szabályos lépések megtételével
juss el a bal felső sarokból a jobb alsó sarokba.
</body>
</html>
```

Lássuk a csatolt JavaScript-fájlt:

```
function lep(){
```

Az előző programhoz hasonlóan itt is kiszedem helyi változókba a szükséges értékeket:

```
  var model=document.getElementById("modell");
  var ins=model.getInstanceDocument("adatok");
  var koor=ins.getElementsByTagName("*");
  var aktx=parseInt(koor[1].firstChild.nodeValue);
  var akty=parseInt(koor[2].firstChild.nodeValue);
  var hovax=koor[3].firstChild.nodeValue;
  var hovay=parseInt(koor[4].firstChild.nodeValue);
  var szinakt=parseInt(koor[5].firstChild.nodeValue);
  var szinhova=parseInt(koor[6].firstChild.nodeValue);
```

Az aktuális és a célmező közötti eltérések abszolút értékben:

```
  var dx=Math.abs(hovax-aktx);
  var dy=Math.abs(hovay-akty);
```

Ha elértem a célmezőt, akkor nyertem:

```
  if(aktx==8 && akty==8){
    alert("Megnyerted a játékot,nincs több lépési lehetőséged!");
    return;
  }
```

Csak az alábbi feltételek teljesülése esetén léphetek a bábuval, ekkor az aktuális pozíció értékeit módosítom:

```
  if(dx>=1 && dx<=2 && dy>=1 && dy<=2 && (dx+dy)==3){
    koor[1].firstChild.nodeValue=hovax;
```

```
      koor[2].firstChild.nodeValue=hovay;
```

Attól függően, hogy a kiindulási és a célmező milyen színű volt, módosítom a mezők alapjául szolgáló képeket:

```
      if(szinhova==0){
        document.images[(hovax-1)*8+hovay-1].src="feherh.bmp";
        koor[5].firstChild.nodeValue=0;
      }
      else{
        document.images[(hovax-1)*8+hovay-1].src="feketeh.bmp";
        koor[5].firstChild.nodeValue=1;
      }
      if(szinakt==0)
        document.images[(aktx-1)*8+akty-1].src="feher.bmp";
      else
        document.images[(aktx-1)*8+akty-1].src="fekete.bmp";
```

Ellenőrzöm, hogy nyertem-e:

```
      if(hovax==8 && hovay==8)
        alert("Gratulálok! Megnyerted a játékot!");
    }
```

Ha nem lehetséges a lépés, mert nem szabályos, akkor nem módosítok semmit, csak befejezem a függvényt:

```
    else{
      alert("Nem lehetséges ilyen lépés a huszár figurával! Válassz másik
  mezőt!");
      return;
    }
```

Az XForms-modell frissítése:

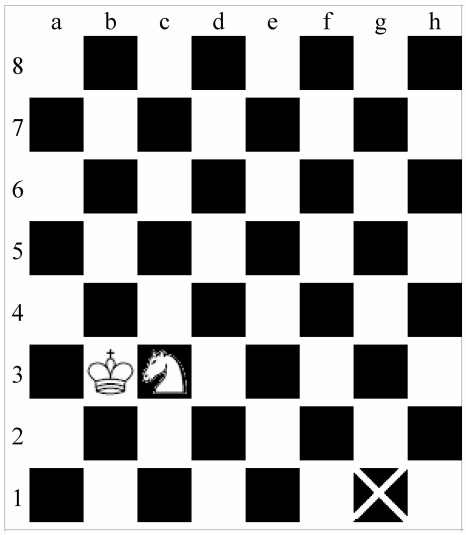
```
    model.rebuild();
    model.recalculate();
    model.refresh();
    return;
  }
```

5.3. Huszár-király játék

- **játék lényege:** van egy király és egy huszár figuránk, és csak azzal a bábuval léphetünk, amelyik éppen ütésben van a másik által, és természetesen csak a neki

megfelelő szabályos sakklépések segítségével. Olyan nem lehetséges, hogy mindkettő üti egymást. Ha esetleg előfordulna az, hogy egyik sem üti a másikat, akkor vége a játéknak. Úgy kell ügyesen lépkednünk, hogy az egyik mindig üsse a másikat.

- **játék célja:** a kijelölt mezőre el kell jutnunk valamelyik bábuval



13. ábra

Láthatjuk a képen a kiindulási állapotot, és a célmezőt, oda kell eljutnunk valamelyik bábuval. A sakktábla megvalósítása megegyezik az előző programban látottéval.

A forráskód:

Szokásos deklarációk, névterek megadása:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xf="http://www.w3.org/2002/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xml:lang="en">
```

Az adatelemek megadása, sorrendben a huszár, majd a király aktuális pozícióinak és ezen pozíciók színének a megadása, aztán azon mező koordinátáinak és színének a megadása, ahová lépni szeretnék, a célmező koordinátái, amely állandó, és az aktuális bábu jelölése (amelyikkel az adott pozícióban lehet lépni):

```
<head>
<xf:model id="modell">
  <xf:instance xmlns="" id="adatok">
    <koordinata>
      <akthx>6</akthx>
      <akthy>3</akthy>
      <aktkx>6</aktkx>
      <aktky>2</aktky>
      <aktszinh>1</aktszinh> <!-- 0 a fehér színt jelent, 1 pedig a
      feketét -->
      <aktszink>0</aktszink>
      <vizhova/>
      <fugghova/>
      <szinhova/>
      <celx>8</celx>
      <cely>7</cely>
      <aktbabu>0</aktbabu> <!-- 0 a huszár, 1 a király -->
    </koordinata>
</xf:instance>
```

Típusok megadása:

```
<xf:bind nodeset="/koordinata/akthx" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/akthy" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/aktkx" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/aktky" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/vizhova" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/fugghova" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/celx" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/cely" type="xsd:decimal"/>
<xf:bind nodeset="/koordinata/aktszinh" type="xsd:binary"/>
<xf:bind nodeset="/koordinata/aktszink" type="xsd:binary"/>
<xf:bind nodeset="/koordinata/szinhova" type="xsd:binary"/>
<xf:bind nodeset="/koordinata/aktbabu" type="xsd:decimal"/>
</xf:model>
```

Stíluslap-információk megadása:

```
<style type="text/css">
  #fekete {background-color: black}
  #elso {text-align: center}
  #cim {text-align: center}
</style>
```

JavaScript fájl belinkelése:

```
<script type="text/javascript" src="huszkir.js"/>
</head>
```

A <body> rész:

```
<body>
  <h1 id="cim">"Huszár és király" játék</h1>
  <table border="1" cellpadding="0" cellspacing="0" align="center"
rules="none">
  …
```

Itt a sakktábla megadása következik, ezt nem részletezném, mivel hasonlóan megy, mint az előző játéknál, különbség csupán a `trigger`-ekre ráhúzott képekben van.

```
  …
  </table>
  <br />
```

Itt a végén még látható egy kiírás, ami megmondja, hogy melyik bábuval lehet az adott körben lépni, ha ez az érték „2”, akkor vége van a játéknak, vagy úgy, hogy nyertünk, vagy úgy, hogy már nem tudunk lépni:

```
  <b>Aktuális bábu (0: huszár/1: király/2: játék vége): </b> <xf:output
  ref="/koordinata/aktbabu" />
…
</body>
</html>
```

A külső JavaScript-fájl:

```
function lep(){
```

A korábbiakhoz hasonlóan a szükséges adatokat helyi változókban eltároljuk:

```
  var model=document.getElementById("modell");
  var ins=model.getInstanceDocument("adatok");
  var koor=ins.getElementsByTagName("*");
  var akthx=parseInt(koor[1].firstChild.nodeValue);
  var akthy=parseInt(koor[2].firstChild.nodeValue);
  var aktkx=parseInt(koor[3].firstChild.nodeValue);
  var aktky=parseInt(koor[4].firstChild.nodeValue);
  var aktszinh=parseInt(koor[5].firstChild.nodeValue);
  var aktszink=parseInt(koor[6].firstChild.nodeValue);
  var hovax=parseInt(koor[7].firstChild.nodeValue);
```

```
var hovay=parseInt(koor[8].firstChild.nodeValue);
var hovaszin=parseInt(koor[9].firstChild.nodeValue);
var celx=parseInt(koor[10].firstChild.nodeValue);
var cely=parseInt(koor[11].firstChild.nodeValue);
var aktbabu=parseInt(koor[12].firstChild.nodeValue);
var dx;
var dy;
```

Ellenőrzi, hogy nem-e nyertük már meg a játékot az előző lépésben:

```
if((akthx==celx && akthy==cely) || (aktkx==celx && aktky==cely)){
  alert("Megnyerted a játékot,nincs több lépési lehetőséged!");
  return;
}
```

Ha vége volt a játéknak, és a játékos nem szerette volna újrakezdeni, akkor ezt írja ki:

```
if(aktbabu==2){
  alert("Vége a játéknak,és nem szeretted volna újrakezdeni sem!");
  return;
}
```

Attól függően, hogy melyik az aktuális bábu, kiszámolja az aktuális mező, és a célmező közti különbséget:

```
if(aktbabu==0){
  dx=Math.abs(hovax-akthx);
  dy=Math.abs(hovay-akthy);
}
else{
  dx=Math.abs(hovax-aktkx);
  dy=Math.abs(hovay-aktky);
}
```

A feltételek teljesülése esetén (amiben benne van, hogy a királyt nem ütheti le) ellép a huszárral:

```
if(aktbabu==0 && (!(hovax==aktkx && hovay==aktky)) && dx>=1 && dx<=2 &&
dy>=1 && dy<=2 && (dx+dy)==3){
  koor[1].firstChild.nodeValue=hovax;
  koor[2].firstChild.nodeValue=hovay;
```

A színeket beállítja a kiindulási és a célmezőn:

```
 if(hovaszin==0){
   document.images[(hovax-1)*8+hovay-1].src="feherh.bmp";
   koor[5].firstChild.nodeValue=0;
  }
```

```
      else{
        document.images[(hovax-1)*8+hovay-1].src="feketeh.bmp";
        koor[5].firstChild.nodeValue=1;
      }
      if(aktszinh==0)
        document.images[(akthx-1)*8+akthy-1].src="feher.bmp";
      else
        document.images[(akthx-1)*8+akthy-1].src="fekete.bmp";
```

Frissíti az XForms-modellt, de előtte még megnézi, hogy nem-e nyertünk:

```
      if(hovax==celx && hovay==cely){
        alert("Gratulálok! Megnyerted a játékot!");
        koor[12].firstChild.nodeValue=2;
        model.rebuild();
        model.recalculate();
        model.refresh();
        return;
      }
```

Módosítja a huszár aktuális pozícióját a helyi változóban, ez a későbbiekben még kell:

```
      akthx=hovax;
      akthy=hovay;
    }
```

A feltételek teljesülése esetén ellép a királlyal (itt is ellenőrzi, hogy a huszárt ne üssük le):

```
    else if(aktbabu==1 && (!(hovax==akthx && hovay==akthy)) && ((dx==0 &&
    dy==1) || (dx==1 && dy==0) || (dx==1 && dy==1))){
      koor[3].firstChild.nodeValue=hovax;
      koor[4].firstChild.nodeValue=hovay;
```

Színek beállítása:

```
      if(hovaszin==0){
        document.images[(hovax-1)*8+hovay-1].src="feherk.bmp";
        koor[6].firstChild.nodeValue=0;
      }
      else{
        document.images[(hovax-1)*8+hovay-1].src="feketek.bmp";
        koor[6].firstChild.nodeValue=1;
      }
      if(aktszink==0)
        document.images[(aktkx-1)*8+aktky-1].src="feher.bmp";
      else
```

```
        document.images[(aktkx-1)*8+aktky-1].src="fekete.bmp";
```

Frissítés, és ellenőrzés, hogy nem-e léptünk a célmezőre:

```
      if(hovax==celx && hovay==cely){
        alert("Gratulálok! Megnyerted a játékot!");
        koor[12].firstChild.nodeValue=2;
        model.rebuild();
        model.recalculate();
        model.refresh();
        return;
      }
```

A helyi változókban a király aktuális pozíciójának a módosítása:

```
      aktkx=hovax;
      aktky=hovay;
    }
```

Szabálytalan lépést akartunk megtenni:

```
    else alert("Nem lehetséges ilyen lépés az aktuális figurával! Válassz
    másik mezőt!");
```

Meghatározzuk, hogy melyik bábu jön a következő lépésben:

```
    aktbabu=aktbabu_beallit(akthx,akthy,aktkx,aktky);
    koor[12].firstChild.nodeValue=aktbabu;
```

Ha egyik bábu sem üti a másikat, akkor vége a játéknak, és nem nyertünk. Ilyenkor van lehetőség újrakezdeni, ekkor mindent visszaállítunk a kiindulási értékre:

```
    if(aktbabu==2){
      alert("Sajnos egyik bábu sem üti a másikat,így vége a játéknak!");
      if(confirm("Szeretnéd újrakezdeni a játékot?")){
        if(aktszinh==0)
          document.images[(akthx-1)*8+akthy-1].src="feher.bmp";
        else
          document.images[(akthx-1)*8+akthy-1].src="fekete.bmp";
        if(aktszink==0)
          document.images[(aktkx-1)*8+aktky-1].src="feher.bmp";
        else
          document.images[(aktkx-1)*8+aktky-1].src="fekete.bmp";
        if(hovaszin==0)
          document.images[(hovax-1)*8+hovay-1].src="feher.bmp";
        else
          document.images[(hovax-1)*8+hovay-1].src="fekete.bmp";
        koor[1].firstChild.nodeValue=6;
```

```
        koor[2].firstChild.nodeValue=3;
        koor[3].firstChild.nodeValue=6;
        koor[4].firstChild.nodeValue=2;
        koor[5].firstChild.nodeValue=1;
        koor[6].firstChild.nodeValue=0;
        koor[12].firstChild.nodeValue=0;
        document.images[42].src="feketeh.bmp";
        document.images[41].src="feherk.bmp";
        model.rebuild();
        model.recalculate();
        model.refresh();
        return;
      }
```

Ha nem kezdjük újra, akkor nem módosítunk semmit:

```
      else{
        koor[12].firstChild.nodeValue=2;
        model.rebuild();
        model.recalculate();
        model.refresh();
        return;
      }
```

XForms-modell frissítése:

```
    }
    model.rebuild();
    model.recalculate();
    model.refresh();
    return;
  }
```

Ez a függvény beállítja az aktuális bábut, ha a huszár az, akkor „0”-t, ha a király, akkor „1”-t, egyébként meg, ha egyik sem üti a másikat, akkor „2”-t ad vissza:

```
  function aktbabu_beallit(hx,hy,kx,ky){
    var elteresx=Math.abs(hx-kx);
    var elteresy=Math.abs(hy-ky);
    if((elteresx==1 && elteresy==0) || (elteresx==0 && elteresy==1) ||
    (elteresx==1 && elteresy==1))
      return 0;
    else if(elteresx>=1 && elteresx<=2 && elteresy>=1 && elteresy<=2 &&
    (elteresx+elteresy)==3)
```

```
    return 1;
  else return 2;
}
```

5.4. Összefoglalás

Az XForms-ban egy viszonylag egyszerűen kezelhető nyelvet ismertem meg. Az alapdolgok elsajátítása véleményem szerint senki számára nem okozhat különösebb nehézséget. Természetesen lehet bonyolítani is, és lehet benne összetettebb szerkezeteket is létrehozni, és használhatunk benne bonyolultabb XPath kifejezéseket is, amelyek kiértékelése egy kicsit lassíthatja a dokumentum feldolgozását, ami szerintem alapjában véve gyorsnak mondható. Láthattuk például a Flexnél, hogy ahhoz létezik fejlesztőeszköz, ahol úgymond rá lehet dobni a felületre a gombokat. Az XForms-hoz tudomásom szerint nincs ilyen eszköz, itt minden kódot be kell gépelni valamilyen szövegszerkesztőben, vagy egy egyszerű XML-szerkesztőben. Én az Architag XRay XML Editort használtam.
Sajnos teljes implementációk még nem léteznek az XForms-hoz, a böngészőkben is egy külön kis kiegészítőt kell feltelepíteni ahhoz, hogy az XForms elemeit tudja értelmezni.

6. Befejezés

A felhasználói felületet leíró nyelvek egyre jobban elterjednek, ezek közül láthattunk néhányat említés szintjén, illetve egyet részletesebben bemutatva. Ezeknek a nyelveknek van jövőjük, hiszen segítségükkel létre tudunk hozni webes és nem webes alkalmazásokat, azok felhasználói felületét. Azt most még nem tudhatjuk, hogy közülük melyik fog győztesként kikerülni, ez még a jövő zenéje. Sok különbség van köztük, amely különbségek szolgálhatnak előnyként, de akár hátrányként is. A Flex például nem ingyenes, de nagyon jól használható a szerkesztője, komplexebb megoldások is megvalósíthatóak vele. Az XAML, mivel a Microsoft terméke, csak azon belül használható, a WPF nyelveként jött létre, ami a Vistában jelent meg. Erre a nyelvre épülve viszont sokféle nyelven lehet a WPF-hez alkalmazásokat fejleszteni. De mivel a Microsoft terméke, így nem hordozható. Az XForms hordozható, de teljes implementáció nincs hozzá, és pluszba még kiegészítők keresgélésével kell töltenünk az időnket a különféle böngészőkhöz.

Annyit mindenesetre kijelenthetünk, hogy ezen nyelvek létezését nem lehet megkérdőjelezni, hiszen a webes alkalmazások, ezáltal a webes felhasználói felületek jelentősége a jövőben mindenképpen nőni fog, mert a web még tartogat számunkra kiaknázatlan lehetőségeket.

7. Irodalomjegyzék

Könyvek:

[1] Bócz Péter-Szász Péter: *A világháló lehetőségei: Interaktív Weblapok készítése.* Budapest, 2003, ComputerBooks.

Online források:

[2] http://www.gtk.org/index.html

[3] http://en.wikipedia.org/wiki/GTK+

[4] http://www.gtk.org/overview.html

[5] http://www.adobe.com/products/flex/technologies

[6] http://en.wikipedia.org/wiki/MXML

[7] http://www.adobe.com/products/flex/whitepapers/pdfs/flex2wp_technicaloverview.pdf

[8] http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language

[9] http://www.xaml.net/

[10] http://www.xulplanet.com/tutorials/xultu/intro.html

[11] http://developer.mozilla.org/en/docs/The_Joy_of_XUL

[12] http://hu.wikipedia.org/wiki/FelhasznÃ¡lÃ³i_felÃ¼let

[13] http://hu.wikipedia.org/wiki/Grafikus_felhasznÃ¡lÃ³i_felÃ¼let

[14] http://www.standardsmode.hu/html-css/xhtml

[15] http://www.w3.org/TR/2002/REC-xhtml1-20020801

[16] http://www.prog.hu/cikkek/?title=Bevezet%E9s%20az%20XML-be

[17] http://www.w3schools.com/xml/default.asp

[18] http://www.w3schools.com/xforms/default.asp

[19] http://www.w3.org/MarkUp/Forms/2003/xforms-for-html-authors.html

[20] http://www.w3.org/MarkUp/Forms/2006/xforms-for-html-authors-part2.html

[21] http://www.w3schools.com/dom/default.asp

[22] http://www.mozilla.org/projects/xforms/samples.html

[23] http://developer.mozilla.org/en/docs/XForms:Form_Troubleshooting

[24] http://www.w3.org/TR/2007/REC-xforms-20071029/

[25] web.conf.hu/2006 Magyarorszagi_Web_Konferencia_2006_Fuzet, PDF-fájl

[26] http://en.wikipedia.org/wiki/XSLT

[27] http://www.inf.unideb.hu/~jeszy/download/xml/xml_10.pdf

[28] http://www.inf.unideb.hu/~jeszy/download/xml/xpath_10.pdf

[29] http://www.inf.unideb.hu/~jeszy/download/xml/xml_nevterek.pdf

[30] http://www.w3.org/TR/2006/REC-xml-20060816/

[31] http://www.w3.org/TR/REC-xml-names/