Web-alapú metanyelvek dokumentumainak validációja szemantikus szabályokkal

Doktori értekezés tézisei

Kálmán Miklós

Témavezető:

Prof. Gyimóthy Tibor
tanszékvezető egyetemi tanár

Informatika Doktori Iskola
Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged
2014

Bevezetés

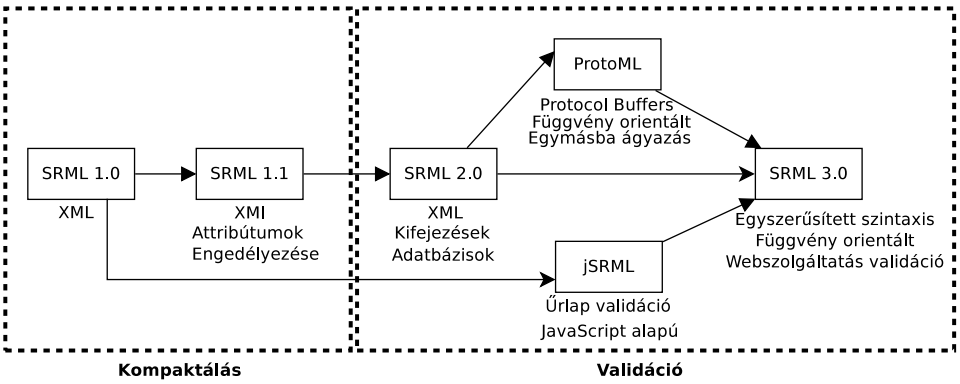
A validáció egyre fontosabb szerepet játszik a folyamatosan növekvő globális információcsere terén. A rendszerek naponta nagyjából 1.000 petabyte adatot osztanak meg egymással. Az adat teljes életciklusa során fontos az adatok integritásának elérése és fenntartása. Az adat validáció fontossága a webes űrlapoktól a dokumentumcserén és az adatbázisok rekordjain keresztül egészen a webszolgáltatások területéig terjed. Az XML dokumentum az információcsere egyik leggyakoribb szöveges formátuma. Ez a szöveges formátum képes hierarchikus kapcsolatokat leírni. Az XML sok nyelv alapját képezi, mivel könnyen kiterjeszthető. Egy korábbi cikkben [13] bemutattuk, hogy miként lehet az SRML metanyelvünk segítségével olyan szemantikus szabályokat definiálni, melyek képesek az XML dokumentumokat kompaktálni. A nyelv kezdeti (1.0, 1.1) verziója az XML állományokat kompaktálta a szemantikus szabályokkal kiszámolható attribútumok eltávolításával.

Tézis rövid címe	**Tézis**	**Publikációk**
XML Dokumentumok validációja	Az SRML 1.0 nyelv kiterjesztése, melynek segítségével az XML dokumentumok validációja és javítása lehetségessé válik.	[12]
Webes űrlapok validációja	A jSRML metanyelv létrehozása, amely képes szemantikus szabályok alkalmazásával validálni és javítani a webes űrlapokat.	[10]
Google Protocol Buffers üzenetek validációja	Létrehozni egy metanyelvet (ProtoML), amely a Google Protocol Buffers üzeneteit képes validálni és kijavítani.	[9]
Webszolgáltatások validációja	A korábbi metanyelvek (SRML 2.0, jSRML, ProtoML) egyesítésével ProtoML) létrehozni egy új SRML 3.0 nyelvet, valamint megoldást találni a webszolgáltatások szemantikus szabályokkal történő validációjára.	[11]

1. táblázat. Tézisek és publikációk

Célul tűztük ki, hogy az SRML 1.0 metanyelvet kiterjesztjük a validáció területére, amely a dolgozat fő irányát adja. Bemutatjuk az SRML nyelv fejlődését további két metanyelv leírásával (jSRML[10] és ProtoML[9]), amelyek a kiinduló SRML leírás párhuzamos ágaiként jöttek létre. Mindegyik megemlített nyelv aktív szerepet játszik a saját területén. A jSRML nyelv webes űrlapok validációs szabályainak leírását célozta meg, a ProtoML pedig a Google Protocol Buffers[5] helyességét képes validációs logikával biztosítani. A dolgozat negyedik validációs területét a webszolgáltatások alkotják. Bemutatjuk, hogy a jSRML és ProtoML metanyelvek kifejlesztése során nyert tapasztalatok miként járultak hozzá az SRML 3.0[11] létrehozásához. A legújabb kiterjesztés segítségével lehetőség nyílik a webszolgáltatások kérelmei és válaszai tartalmának validálására is.

Az SRML nyelv kiterjesztéseinek magas szintű áttekintését a 1. ábra tartalmazza. A dolgozat négy tézisét az *1. tábla* foglalja össze.



1. ábra. Az SRML nyelv fejlődése és alkalmazási területei

1. XML Dokumentumok validációja

Tézis: Az SRML 1.0 nyelv kiterjesztése, melynek segítségével az XML dokumentumok validációja és javítása lehetségessé válik.

A dolgozat első tézise az XML validáció. Az XML dokumentumok struktúrájának leírásához a leggyakrabban alkalmazott módszer a DTD[15] leírások, illetve XSD[16] sémák használata. Az utóbbi sokkal fejlettebb, mivel képes leírni a struktúrális elemeket, illetve azok típusait és engedélyezett elemeit meghatározni. Az XSD sémák nagyon hatékonyak, viszont egy fontos leíró aspektus hiányzik belőlük: a tartalom validáció. Ahhoz, hogy az XML dokumentumok tartalmi korlátozásait leírjuk, külső rendszerek megoldásait kell használni. Az SRML 1.0 nyelv kiterjesztésével egy szabály-orientált megoldást adunk az XML validációra. A *2. táblázat* bemutatja az SRML verziók közötti főbb különbségeket.

Tulajdonság	SRML 1.0	SRML 2.0
Terület	Kompakció/Dekompakció	Validáció/Hibajavítás
Szabály hivatkozási szint	Attribútumok	Elem és Attribútum
Alkalmazhatóság	XML Dokumentumok	XML és Adatbázisok
Szabályok alapja	Attribútum Nyelvtanok	AG és XPath
Szabály komplexitás	Bonyolult	XPath-el leegyszerűsítve
Szabályok kifejezései	Nagy méret és komplexitás	Egyszerűsített, belső kifejezés motor
Szabály függőség és tárolás	DTD és külső SRML állomány	XSD-be injektálva

2. táblázat. SRML 1.0 és 2.0 közötti főbb különbségek

Az új SRML verzió számos újítást tartalmaz, melyek közül az alábbiakat célszerű kiemelni:

XPath támogatás: Az XPath segítségével könnyebbé válik az XML attribútumokra és elemekre való hivatkozás. Korábban a hivatkozásokat az Attribútum Nyelvtanoknál ismert módon kezelték, amely bár sokoldalú leírást tett lehetővé, bonyolult definíciókat eredményezett.

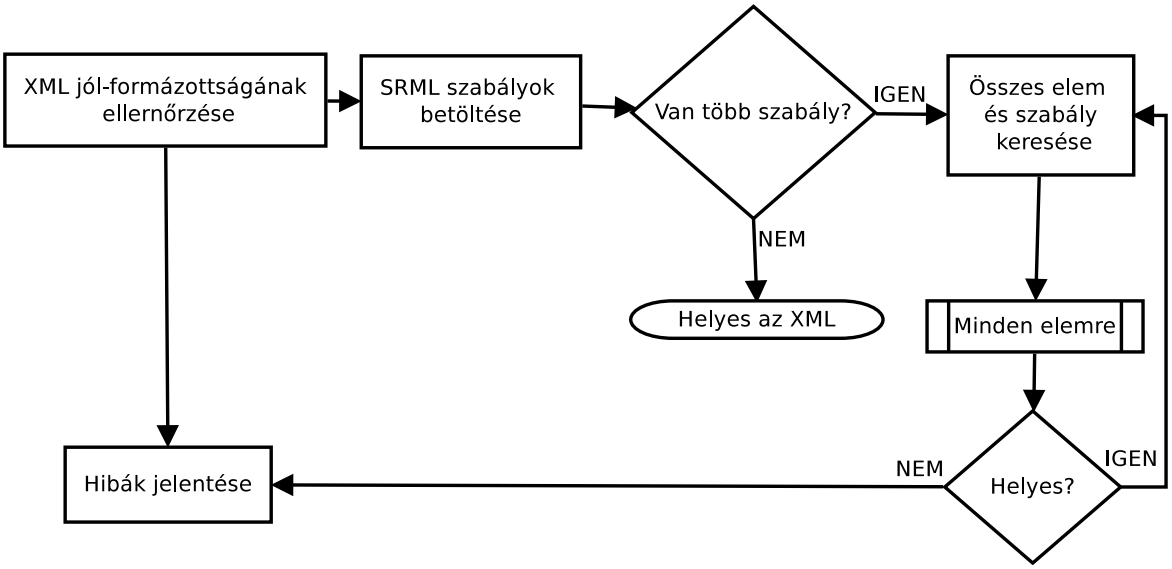
Numerikus kifejezések: Az új formátum lehetőséget biztosít arra, hogy numerikus kifejezéseket használjunk a szabály definíció során. Ez jelentősen leegyszerűsíti a szabályok leírását és olvashatóbb formát biztosít.

Elem és attribútum hivatkozások: Korábban csak attribútum hivatkozást engedélyezett a nyelv. A kiterjesztésnek köszönhetően most már mindkét entitás típusra hivatkozhatunk.

Kontextuson belül több szabály definíció engedélyezése: Ennek az új jellemzőnek köszönhetően több szabályt lehet definiálni ugyanazon kontextuson belül. Ez a validációhoz rendkívül fontos funkcionalitás, mivel lehetőség nyílik több, akár a környezettől függő szabály definiálásra. Ilyen esetben a validáció akkor lesz sikeres, ha legalább egy szabály teljesül a vizsgált elemre, vagy attributumra.

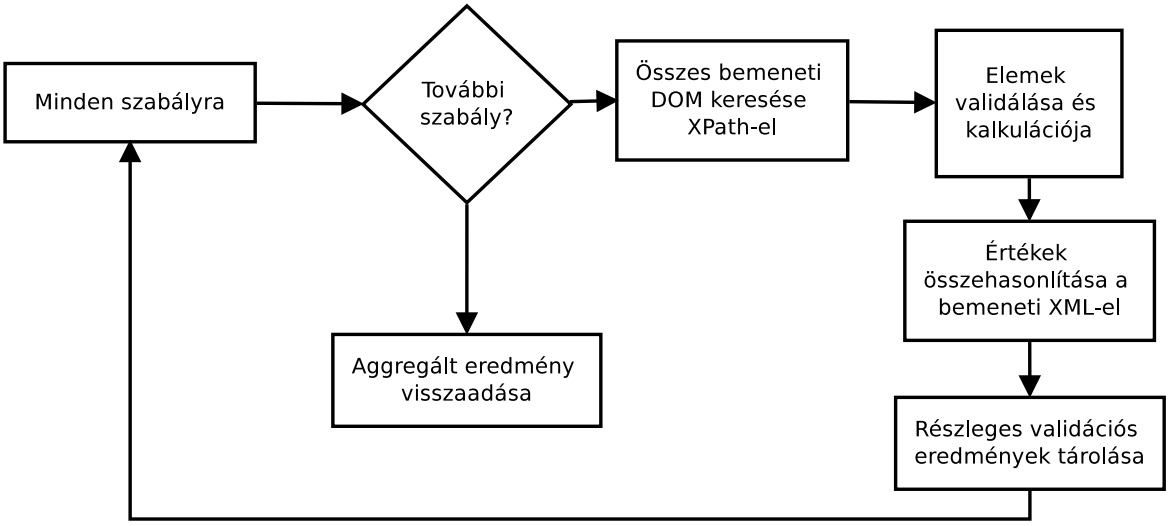
Szabályok beágyazása XSD file-ba: Lehetőség van a szabályokat a validációs XSD dokumentum `appinfo` részében definiálni. Ennek köszönhetően a tartalmi és struktúrális validációt egy helyen lehet leírni.

Az SRML nyelv kiterjesztése során egy olyan megoldást akartunk biztosítani, amely képes a strukturális validációs eljárásba beépülni. Mivel az XSD sémákat strukturális validációra használják, így ideális fejlesztési terület volt. Az XSD `appinfo` részének segítségével egy nem tolakodó módszert tudunk biztosítani az SRML szemantikus szabályok tárolására. Ennek segítségével az XSD dokumentumok képesek lesznek mind a strukturális, mind a tartalmi validációs szabályokat leírni. A *2. ábra* bemutatja az XSD és SRML validációs folyamatát. Az SRML 2.0 nyelv funkcionalitásának bemutatására létrehoztunk egy validációs motort, az `SRMLXsdTool`-t.



2. ábra. XML validációs folyamata XSD és SRML segítségével

Az SRML szabályok hatékonyan alkalmazhatóak az adatbázis rekordok validációjára is. Bemutattunk egy módot, mely segítségével lehetőség nyílik az SRML szabályok adatbázisba való beszúrására és rekordjainak triggereken keresztül történő validálására. Ezt a H2[2] adatbázis segítségével értük el, amely képes Java osztályokat használni adatbázis triggerekként. A validációs motorunkat is Java nyelven készítettük el, így képesek voltunk olyan trigger osztályokat implementálni, amelyek bizonyos adatbázis műveletek során végrehajthatóak. A legnagyobb kihívás ennek során az volt, hogy miként tudjuk az adatbázis rekordokat DOM[1] faként ábrázolni. A DOM fák az XML dokumentumok hierarchikus reprezentációi. A rekordokat kilapítottuk és a szabály struktúrát kibővítettük a táblák közti kapcsolatok leírásával. Ez a bővítés hasonló az idegen kulcsok működési elvéhez. Ezen kulcsok segítségével a rekordokat összekapcsolhatjuk és az oszlopaikat attribútumként használhatjuk. Az adatbázis rekordok validációs folyamatát a *3. ábra* mutatja.



3. ábra. Adatbázis rekordok SRML szabályokkal történő validálációja

Az SRML 2.0 nyelv lehetővé teszi a hibajavítást is, továbbá lehetőséget biztosít az XML dokumentumok tartalmának szemantikus szabályokkal történő korrekciójára. Amikor a *javítás* üzemmódot használjuk, akkor a validációs motor kiszámolja a vizsgált elem elvárt értékét és helyettesíti azt, amennyiben a validáció hibát eredményezett. A dokumentumok javításának ezt a hatékony módszerét a jSRML és a ProtoML ágakban is alkalmazzuk.

1.1. Tézis összefoglalása és saját eredmények

- Bemutattuk, miként lehet kiterjeszteni az SRML 1.0 nyelvet a validáció terére. Az eredeti nyelv specifikációja az XML dokumentumok kompaktálását célozta szemantikus szabályokkal.
- Az új formátum integrálódik az XSD validációs sémába, amely egy hordozható megoldást hoz létre. Ezen megoldásban mind a strukturális, mind a tartalmi validációs logika egy közös dokumentumban jelenik meg.
- Az új nyelv támogatja az XPath hivatkozásokat és a numerikus kifejezéseket, melyek segítségével jelentősen leegyszerűsödnek a szabálydefiníciók.
- A kiterjesztés egy további jelentős újítást is bemutat: az adatbázisok rekordjainak validációját szemantikus szabályokkal. Ez a kiterjesztés az adatbázisok triggerei segítségével azok műveletei során képes a rekordok tartalmát manipulálni.
- Bemutattuk, hogy miként lehet a kiterjesztés segítségével az XML dokumentum hibás értékeit kijavítani.

Az SRML validációra való kiterjesztése elsősorban az én kutatásom eredménye, melyet a [12] publikáció részletez.

2. Webes űrlapok validációja

Tézis: A jSRML metanyelv létrehozása, amely képes szemantikus szabályok alkalmazásával validálni és javítani a webes űrlapokat.

A dolgozat második tézise a webes űrlapokat érinti. Az Internet behálózza az életünket. Napról napra több ember használja a világhálót információszerzésre, szórakozásra és munkavégzésre, egyre több tevékenység érhető el online (pl. bevásárlás, adóbevallás... stb). Ennek következtében az adat validáció kulcsfontosságú szerepet tölt be mindennapjainkban. A felhasználók információt cserélnek egymással és fontos, hogy az adat helyes legyen és ne sérüljön. A felhasználók közötti kommunikáció és adatok bevitele webes űrlapok segítségével történik. Ezen űrlapok mezőit a felhasználók kitöltik és továbbítják egy szervernek feldolgozásra, mely ezután feldolgozza az adatokat, műveleteket hajt rajtuk végre, majd visszaküldi a felhasználónak. Ezek a webes űrlapok egyszerű belépési űrlapoktól egészen az online adóbevallásig terjednek, amelyek bizalmas információt forgalmaznak. Sajnos ez a leggyengébb láncszem a rendszerben, amit sok hacker ki is használ. Az űrlapokat HTML[14] oldalak tartalmazzák, amelyek hasonló formátumúak, mint az XML dokumentumok, s emiatt a DOM modell szintén alkalmazható. Ily módon az űrlapok ideálisnak minősülnek a szemantikus szabályokkal történő validációra.

Kiterjesztettük az SRML 1.0 verziót egy új metanyelvre, a jSRML-re, amely képes az űrlapokat validálni. Szükséges megjegyezni, hogy ez a nyelv párhuzamosan jött létre az SRML 2.0 metanyelvvel, azaz nem abból terjesztettük ki az új nyelvet. A jSRML nyelvi elemei és szintaxisa hasonlítanak az SRML 2.0-éra, viszont az új nyelv teljesen más eszköztárral rendelkezik. Az új jSRML nyelv segítségével a felhasználók képesek lesznek SRML szabályokat definiálni a webes űrlapok mezőire, leírni azok kapcsolatát, tartalmuk korlátozásait, illetve formai követelményeit. A validációs motort bármely HTML oldalban használhatjuk. Ehhez mindössze hivatkozni kell a script állományra a dokumentumban és a szabályokat beilleszteni. Ezzel a megközelítéssel nem akadályozzuk a HTML tartalmat felesleges JavaScript kóddal. A jSRML szabályokat azon mezők alá kell beilleszteni, amelyekre a validációs szabályok vonatkoznak, az ezt követő feladatokat a validációs motor automatikusan végzi. A validációs motor számos validáció fajtát támogat, amelyeket a *3. táblázat* tartalmaz.

A jSRML nyelv létrehozása során vettük az eredeti SRML 1.0 nyelvet és a kompaktáló motort (`SRMLTool`) és újraépítettük JavaScript és jQuery segítségével, hogy kimagasló böngésző teljesítményt biztosítson. Úgy döntöttünk, hogy a kiterjesztést jSRMLnek nevezzük és a validációs motort `jSRMLTool`nak, hogy ezzel is jelöljük a JavaScript kapcsolatot. Korábban az SRML szabályokat külön állományban tároltuk, amelynek voltak előnyei és hátrányai. Az előnye ennek a megközelítésnek az, hogy az összes szabály egy helyen volt, viszont nehezebb megkeresni, melyik elem kontextusra vonatkozott a szabály. A jSRML-ben lehetőség van az adott mezők mellé beszúrni a rájuk vonatkozó szabályokat, de a nyelv támogatja a külső állományokban történő definiálást is.

A jSRML másik nagy előnye, hogy nem tolakodó, azaz nem akadályozza az állomány tartalmát és átláthatóságát. Használatához mindössze be kell illeszteni a script hivatkozást. Abban az esetben, ha a validációs szabályokat változtatnunk kell, nem kell a kódot módosítani, csak a szabályokat, ezzel is csökkentve a hibalehetőséget.

A jSRML motor képes kijavítani a mezők értékeit, ha a szabály definíció tartalmazza ezt az utasítást. Ez egy hatalmas előny a többi szabály vagy JavaScript alapú validátorhoz képest, mivel ilyenkor a motor képes kijavítani a hibát és a beküldést sikeressé teheti.

Típus	Indítás	Feldolgozás	Validációs logika	Előny	Hátrány
Szerver Oldal	Űrlap Beküldés	Szekvenciális	Böngészőbe visszatérés az eredmények megjelenítésére	A Validációs logika rejtett a felhasználó elöl	A Validációs szabályok változása szerver oldali változást igényelnek
Kliens Oldal	OnClick elfogás	Kliens oldalon	Böngészőben JavaScripttel történő megjelenítés	Gyors, mivel nem küld adatokat a szervernek	A Validációs logika nyilvánosan elérhető
Valós Idejű	Mező változás	Bármely	Közvetlen, Kliens és/vagy Szerver validációs hivás	Mezők értékeinek valós időben történő kiertékelése beküldés előtt	Nagyobb forgalomigény, nehezebb frissíteni
Hibrid	Mező változás és Beküldés	Bármely	Közvetlen szerver hivás nélkül	Két lépésű validáció, szűrés szerverhez történő beküldés előtt	Bonyolultabb implementálni és fenntartani

3. táblázat. Űrlap Validació típusok

Kiváló példaként említhető a helyesírás ellenőrzés, ahol az űrlap beküldése előtt validációs szabályok ellenőrzik az adatbevitel helyességét. Emiatt a jSRML sokkal szélesebb körben alkalmazható, mivel a fejlesztők kiterjeszthetik a motort saját függvényeikkel.

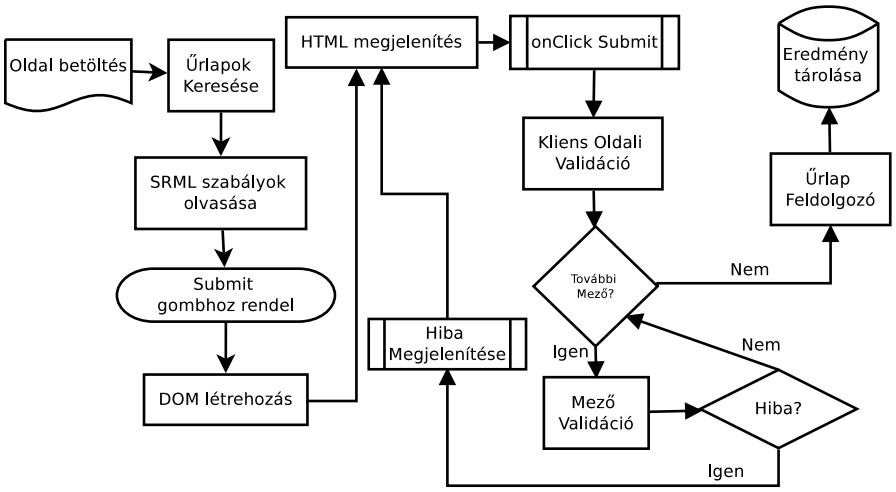
A jSRMLTool motor a korábban említett négy validációs mód mindegyikét támogatja (*Kliens*, *Szerver*, *Valós-idejű*, *Hibrid*), és kimagasló teljesítményt nyújt, mivel a felhasználót nem korlátozza egy megoldásra. A következő rész összefoglalja a jSRMLTool különböző validációs módjait:

- `Kliens-oldali:` Ebben a módban a validációt a beillesztett jSRML.js állomány segítségével végezzük. A szabályokat XPath feltételek segítségével nyerjük ki a HTML állományból. Minden in-line (sorközi) szabályt kommentekben tárolunk, amelyek az [SRML] szóval kezdődnek. A submit gomb onClick eseménykezelőjébe beszúrunk egy hook-ot (csatlakozási hivatkozás). Ennek segítségével a motor validálja a mezőket, amikor lenyomtuk a gombot. Ha sikeres volt a validáció (vagy ki lett javítva az elvárt értékek segítségével), akkor az űrlap mezőinek tartalmát továbbítja az eredeti célpontra, amit az „action" attribútum határoz meg. A *4. ábra* bemutatja a *Kliens-oldali* validációt.

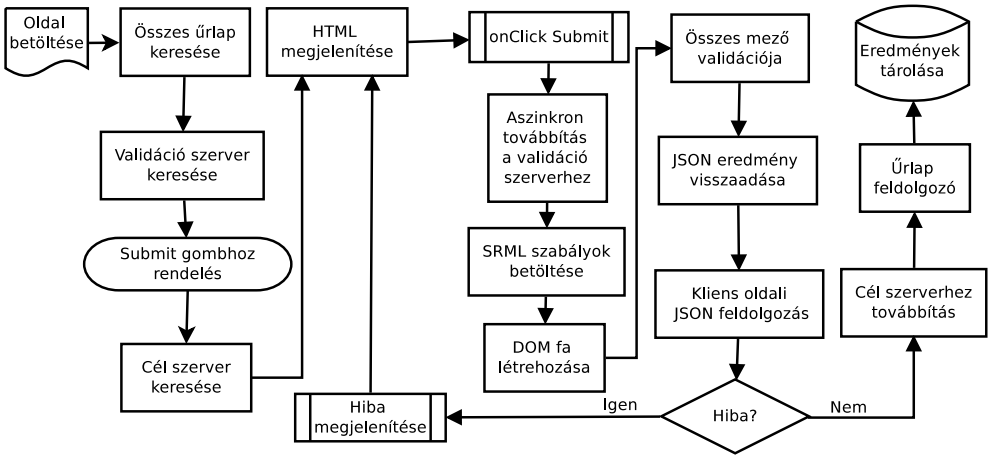
- `Szerver-oldali:` A motor a szerver-oldali módot egy külön servlettel kezeli (jSRMLToolServlet). Ez a servlet egy egyedi azonosító (UID) segítségével azonosítja be az űrlapokat. Ennek segítségével számos űrlapot képes lekezelni, amelyek külön területekhez tartoznak. A validációs folyamat hasonló a *Kliens-oldal*hoz, viszont itt a mezők tartalmát és az egyedi azonosítót is továbbítja a rendszer a servletnek. A servlet elvégzi a validációt/javítást és visszatér a válasszal. A *Szerver-oldali* validációt az *5. ábrán* láthatjuk.

- `Valós-idejű és hibrid:` Minden szabálynak van egy *„method"* attribútuma. Ez nem egy kötelező attribútum és az alapértelmezett értéke *„standard"*. Amikor egy szabály ennek az attribútumnak *„focus"* értéket ad, akkor egy hook-ot telepít a rendszer leírt mező *onBlur* eseménykezelőjébe. Ilyenkor a validáció akkor fog lefutni, ha a szabály által hivatkozott mező elveszíti a fókuszt. Az attribútum harmadik értéke *„real-time"* (valós-idejű). Ebben az esetben a hook

a *keyDown* (billentyű leütés) eseménykezelőjébe települ, amikor is a mezőn belüli minden billentyű leütés során lefut a validációs szabály. Ezt például egy jelszó mező megfelelő hosszának vizsgálatához használható.



4. ábra. Kliens-oldali jSRML



5. ábra. Szerver-oldali jSRML

Amint azt a *Szerver-oldali* validációs módnál említettük, a Java Servlet[8] technológiával implementáltuk a jSRML motor egy szerver oldali verzióját, mely segítségével az űrlapokat aszinkron módon képes validálni. A szolgáltatás forráskódja nem változik a szabályok változtatása esetén. Erre azért nyílik lehetőség, mert a szabályokat a szerver eltárolja egy adatbázisba és az egyedi azonosító segítségével megkeresi, majd az alapján végzi el a validációt. A jSRMLToolServlet akár több ezer különböző űrlapot képes egyszerre validálni, amennyiben fel lettek töltve a hozzá tartozó szabályok. Ezt egy külső szolgáltatásként is lehet üzemeltetni, ami akkor validál űrlapokat, amikor szükség van rá.

A `jSRMLToolServlet` képes validációs szabályokat gépi tanulással előállítani.Ennek a segítségével az űrlapok tulajdonosai javaslatokat kaphatnak a lehetséges validációs szabályokra és ezek alapján építhetik/finomíthatják azokat. A tanuláshoz szükséges adatokat egy az űrlap submit eseményére helyezett hook segítségével szerzi meg a jSRMLToolServlet. Ilyenkor a mező adatai a servlethez továbbítódnak.

Ez hasonlóan működik a szerver-oldali validációhoz, viszont itt nincsenek tárolt szabályok, csak a mezők értékeit gyűjti és érték-név párokként tárolja az egyedi űrlap azonosítóval együtt. A tanuló modul ezeket az értékeket fogja kiértékelni és ez alapján tesz javaslatot az esetleges jSRML szabályokra. Az adatok begyűjtését követően a vezérlés és az eredeti adatok továbbításra kerülnek a tényleges feldolgozó felé, ez egyben azt is jelenti, hogy az űrlap működését nem akadályozza a rendszer, csak elfogja a forgalmat, lementi és továbbítja.

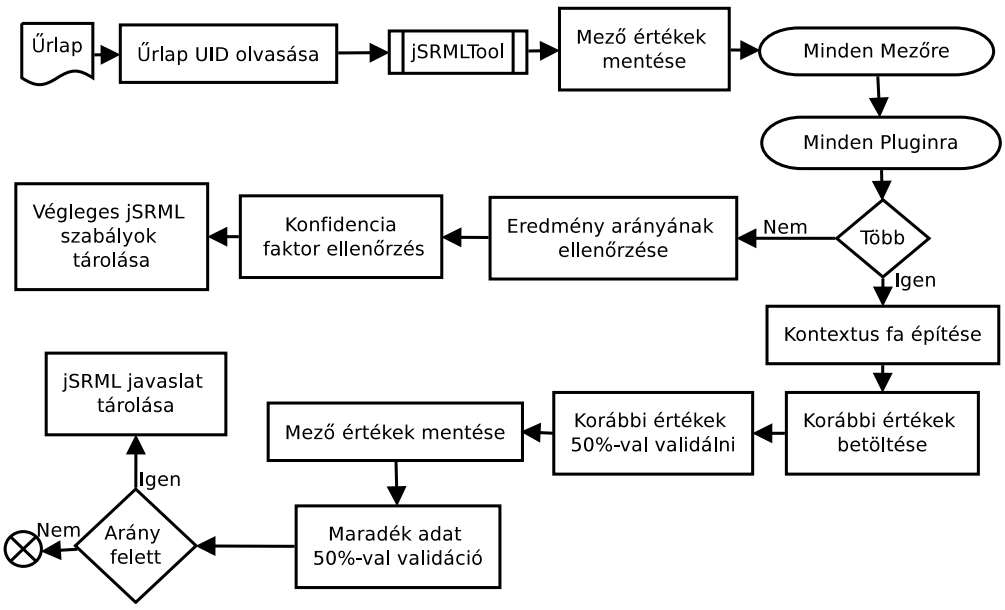
A tanuló modul számos plugin-t (bővítményt) használ az űrlapokból nyert adatok feldolgozásához és a szabályok alakításához, ez pedig egy folyamatos tanulási műveletet eredményez. A motor a tanulást a nyert adatok 50%-án végzi, majd a kapott eredményeket a fennmaradó példákon kipróbálja. Ez azért fontos, hogy ne illessze túl a szabályokat a bevitt mezők értékeire (generalizálás). Jelenleg az alábbi pluginokat tartalmazza a rendszer: *jpFormat*, *jpLength*, *jpCopyContent*, *jpRelationship*, *jpRange*, *jpPredefinedName*, *jpRegExp*. Minden pluginnek van egy konfidencia faktora és egy cél aránya, amit a rendszer adminisztrátora állít be. Ha egy plugin konfidencia faktora magas, akkor minden olyan esetben, amikor a plugin átlépi a célarányt, egy szabályt fog generálni. Minden plugin eredményét kipróbálja a rendszer a példák megmaradó 50%-án, majd ezt követően visszaad egy értéket, ami megmondja hány százalékban fedte le az új szabály a példákat. Lehetséges, hogy több plugin ad javaslatot ugyanarra a mezőre. Ilyen esetekben a rendszer a legmagasabb konfidencia értékű plugin-t fogja választani, ami átlépte a célarányt, ami pedig azt a legkisebb arányt jelöli, amely alatt a rendszer nem fogad el szabály javaslatot.

A pluginok megtartják a hisztorikus (korábbi) mező értékeket, hogy a tanulás során felhasználhassák őket. A tanuló modul végigmegy az összes pluginon és összegyűjti a részleges jSRML szabály javaslatokat. Amint minden plugin lefutott, az eredmények súlyozásra és kiértékelésre kerülnek, a létrejött szabályokat pedig lementi a rendszer. A *6. ábra* bemutatja, hogyan működik a tanulási folyamat. A hatékonyság növelése érdekében célszerű a tanulást felügyelt példákkal kezdeni. Ezen folyamat során a tulajdonos "megtanítja" a motort, hogy melyek a mezők helyes értékei. Előfordulhat, hogy a korábbi űrlapok mezőinek értéke nagyobb mennyiségben elérhető. Ilyen esetekre a rendszerünk lehetőséget biztosít az értékek CSV állományból történő betöltésére. A tanuló modul könnyen bővíthető, moduláris felépítésű, így könnyű új és hatékonyabb pluginokkal növelni a tanulás eredményességét.

A pluginok hatékonyságát program segítségével kiértékeltük és egy valós példára is alkalmaztuk. Amennyiben a tanuló példák pozitívak és nem tartalmaznak zajt, a tanuló modul hatékonysága konvergál a 100%-hoz. A *4. tábla* mutatja a tanuló modul kísérleti eredményeit. A táblázatban látható, hogy minél több pozitív példát kap a rendszer, annál hatékonyabban képes döntést hozni. A *jpRelationship* pluginunk képes kapcsolatokat találni a mezők között, így ezt egyfajta kezdetleges adatbányászatnak is lehet tekinteni.

2.1. Tézis összefoglalása és saját eredmények

- Létrejött a jSRML metanyelv, amely képes szemantikus validációs szabályokat definiálni a webes űrlapok számára. Az új nyelv jól bővíthető és lehetőség van külső függvények használatára is.
- A megoldás nem tolakodó és képes az űrlapok kódjába beszúrni a validációs szabályokat.
- A nyelv segítségével lehetőség nyílik kontextus-függő szabályokat definiálni, amely hasznos eszköz lehet a feltételes érték validáció terén.



6. ábra. jSRMLTool tanulási folyamata

Vizsgált Plugin	Összes Példa	Hibák Száma	Sikeresség Aránya
jpLength	100	17	83.00 %
jpLength	200	7	96.50 %
jpLength	300	2	99.33 %
jpLength	400	0	100.00 %
jpRange	100	72	28.00 %
jpRange	200	85	57.50 %
jpRange	300	97	67.67 %
jpRange	400	81	79.75 %
jpRange	500	63	87.40 %
jpRange	600	59	90.17 %
jpRange	700	45	93.57 %
jpRange	800	34	95.75 %
jpRange	900	28	96.88 %
jpRange	1,000	11	98.90 %
jpRegExp	100	98	2.00 %
jpRegExp	200	186	7.00 %
jpRegExp	300	198	34.00 %
jpRegExp	400	146	63.50 %
jpRegExp	500	90	82.00 %
jpRegExp	600	48	92.00 %
jpRegExp	700	22	96.85 %
jpRegExp	800	12	98.50 %
jpRegExp	900	6	99.33 %
jpRegExp	1,000	2	99.80 %

4. táblázat. A bővítmények hatékonysága fokozatos pozitív példák függvényében

- A jSRML szabályok képesek a hibás mezők értékeit kijavítani a szabályaik segítségével. Ez potenciálisan sikeressé teheti az űrlap beküldési folyamatát.
- A `jSRMLTool` validációs motor mind a négy validációs üzemmódban használható (Szerver-oldali, Kliens-oldali, Valós-idejű, Hibrid)
- A validációs motor mellé egy servlet alapú implementáció is kifejlesztésre került, amely képes számos domain különböző űrlapjainak VaaS-aként (Validáció mint Szolgáltatás) működni.

- A validációs servlet képes az űrlapok értékeit elfogni és azokat tárolni. Ezt eredményeket ezt követően a tanuló modul számos gépi tanulású algoritmussal ellátott plugin (bővítmény) segítségével kiértékeli, majd validációs szabályokat javasol. A tanuló modul képes a mezők és értékei közti összefüggéseket feltárni és ezzel egyfajta adatbányászati eszközként is alkalmazható.

A jSRML kifejlesztése és alkalmazása a webes validációra teljes mértékben az én kutatásom eredménye, amely a [10] folyóiratban került publikációra.

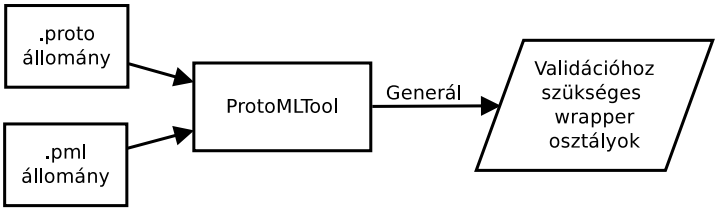
3. Google Protocol Buffer üzenetek validációja

Tézis: Létrehozni egy metanyelvet (ProtoML), amely a Google Protocol Buffers üzeneteit képes validálni és kijavítani.

A dolgozat bemutatja a ProtoML[9] metanyelvet is, amely képes a Google Protocol Buffers[5] (PB) üzeneteit validálni. A PB üzenetek bináris formátumúak, amely külön kihívást jelentett az eddigi szöveg-alapú formátumokhoz képest. A bináris formátumok jelentősen kisebb méretűek a szöveg-alapú formátumoknál, emiatt több adatot lehet ugyanakkora méretben továbbítani. Ez az tulajdonság azonban azzal is jár, hogy a formátum le van fixálva és nehezen bővíthető. A legtöbb bináris formátum előre definiált mezőkkel rendelkezik (hasonlóan a C nyelvben ismert struct struktúrákkal). Ezeknek a formátumoknak gyakran nincs validációs sémájuk, és általában nincs mód arra sem, hogy a mezők közötti kapcsolatokat, vagy azok tartalmi követelményeit leírjuk. Az egyetlen validáció, amit nyújtanak, az a mezőik típusának, nevének és esetleges értékhalmazának definiálása (pl.: ENUMok). A validáció általában a programozókra hárul ilyenkor (ritkán tartalmazza a nyelv maga).

A Google Protocol Buffers-ra azért esett a választás, mert hatékony, bővíthető és számos programozási nyelvet támogat. Sajnos a PB üzenetek validációja nem volt része a nyelv specifikációjának így hasonló hiányosságai vannak, mint a többi bináris formátumnak. Az új ProtoML metanyelv XML alapú és függvények segítségével bővíti ki leíró képességét. A nyelv a PB üzenetek mezőinek értékére nyújt validációs szabályokat (a `.proto` állományok felhasználásával). A ProtoML szabályok képesek egy mezőre több szabályt definiálni a kontextusuktól (környezetük) és értéküktől függően. Az XPath segítségével az üzenet különböző mezőit érjük el. A nyelv bővebb kontextust is képes használni a rendszer oldalon bufferezés segítségével (több üzenetet egy kontextusba helyez és ezzel egy nagyobb DOM fán végzi a műveleteit). A ProtoML szabályok képesek definiálni, hogy mit tegyen a validáló motor validációs hiba esetén (*figyelmezetés*, *kilépés*, *figyelmen kívül hagyás*, *javítás*). A „javítás" üzemmód jelzi a validációs motornak, hogy javítsa ki a mező értékét az elvárt érték felhasználásával abban az esetben, ha a validáció sikertelen volt.

A ProtoML nyelvet felhasználva létrehoztunk egy validációs motort ProtoMLTool néven. Ezt a motort Javaban írtuk és mint library-t is lehet használni. A motort a Spring Framework segítségével készítettük. Ez az *Exp4j* könyvtárat használja a kifejezések kiértékelésére. A DOM fa műveleteket a *JDOM*[7] végzi. A motor segítségével lehetőség nyílik a `.proto` állomány és `.pml` (szabály állomány) kombinációjával wrapper kódot készíteni. A generált wrapper kód később már nem használja a `.pml` állományt és le lehet fordítani a protoc által generált Java állományokkal együtt. Erre úgy van lehetőség, hogy a ProtoML szabályokat a rendszer átfordítja függvényhívások sorozatára, amelyet egy statikus osztályba beszúr, s ezzel közel natív teljesítményt ér el. A motornak van egy csatolt üzemmódja is, amely során nem fordítja le a szabályokat, hanem futás közben a `.pml` állomány segítségével validálja a PB üzeneteket. A *7. ábra* mutatja a ProtoMLTool wrapper osztály generálásának folyamatát.



7. ábra. ProtoMLTool wrapper generálás

A validációs folyamatot a *8. ábra* szemlélteti. Az osztály, ami a PB üzenet feldolgozását végzi, be kell, hogy illessze a generált PMLValidator osztályt (ivy dependenciaként). Amikor megérkezik az üzenet, akkor a `PMLValidator.Validate` függvényt meg kell hívni az aktuális Message objektummal, hogy elvégezzük a validációt. A statikus metódus, amit a `.proto` és `.pml` állományokból generál a rendszer, mindig a legfrissebb generált *Message* osztályokat fogja használni. Erre azért van szükség, mivel a PB osztályok generálásánál egyedi típusok és Enum-ok jönnek létre, amelyeket a motor használni fog.

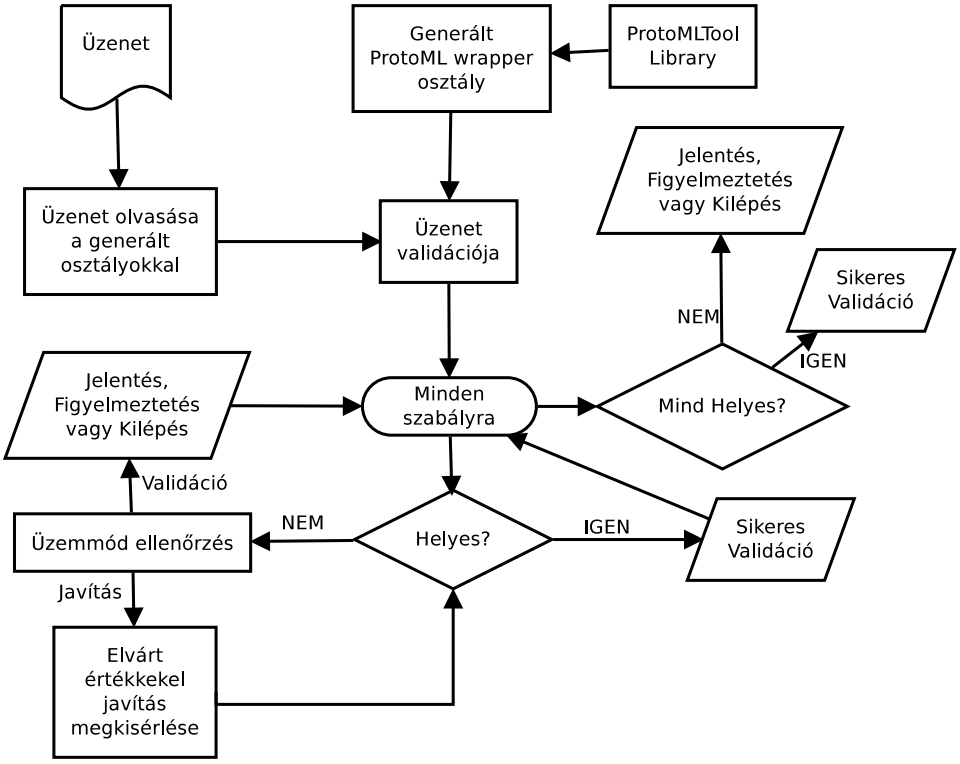
A generált `PMLValidator` osztály egy *HashMap*-ben tárolja le a mezők XPath útvonalát és a hozzá tartozó *validációs leíró bejegyzést*. Ez a bejegyzés tartalmazza a metódusok nevét (amit reflection segítségével fog meghívni a motor) és a hozzátartozó szabály paramétereit (pl.: validálás/javítás módja). A bejövő *Message* objektum feldolgozása során, a motor az üzenetet átalakítja egy DOM fává és a megfelelő metódusokat meghívja rajta. A motor megkeresi a HashMap-veb a mezők XPath útvonalát és megnézi, hogy tartozik-e hozzá szabály bejegyzés. Mivel minden függvény implementációját Java kódként tartalmazza a motor, a szabályok átalakított formátuma *reflection* segítségével hívható lesznek.

Amennyiben a „javítás" üzemmód be van kapcsolva, a motor elsőként azt vizsgálja, hogy sikeresen validálhatók-e a mezők. Amennyiben igen, akkor a validáció sikeresen lezárul. Ha nem, abban az esetben a motor megpróbálja kijavítani a hibákat az elvárt érték segítségével. A motor háromszor kísérli meg a validációt, miután az első validációs hiba miatt javítást végzett a mezőn. Ha a harmadik kísérlet után sem lehet validálni a mezőt, akkor a motor visszatér a validációs hibával.

3.1. Tézis összefoglalása és saját eredmények

- A kifejlesztett ProtoML metanyelv képes a Google Protocol Buffers üzeneteit szemantikus szabályokkal validálni és kijavítani.
- A nyelv függvény-orientált megközelítést használ, mely során egymásba ágyazhatók a függvények. Ez jelentősen kisebb szabály méretet eredményez a korábbi SRML szabályokhoz képest.
- A ProtoMLTool validációs motor a `.proto` leíró állomány és ProtoML szabályokat felhasználva képes Java validációs kódot generálni, amelyet beleilleszthetünk a meglévő kódba. Ez közel natív teljesítményt nyújthat a Google Protocol Buffers validációjára.
- A validációs motor képes külön is futni és a szabályokat futási időben alkalmazni.

A ProtoML nyelv és funkcionalitása teljes egészében kutatásom eredményeit képezik, amelyeket a [9] publikáció részletez.



8. ábra. ProtoMLTool validációs folyamata

4. Webszolgáltatások validációja

Tézis: A korábbi metanyelvek (SRML 2.0, jSRML, ProtoML) egyesítésével létrehozni egy új SRML 3.0 nyelvet, valamint megoldást találni a webszolgáltatások szemantikus szabályokkal történő validációjára.

A dolgozat utolsó tézise a webszolgáltatások validációjával foglalkozik. Az elmúlt időszakban paradigma váltás következett be a szoftveres architektúrák terén. Míg korábban újra írták a szolgáltatásokat, manapság inkább az újrafelhasználás vált mérvadóvá, elsősorban költségtakarékossági okok miatt. A webszolgáltatások áthidalják a földrajzilag erősen osztott rendszerek kommunikációjának nehézségeit, és egy platform-független kommunikációt hoznak létre. Az egyik nagy előnye a webszolgáltatásoknak az, hogy a felhasználó rendszernek nem kell tudnia, hogyan jött létre a kért adat és milyen forrásokból származik. A felhasználó rendszer saját üzleti logikát implementálhat a kapott adatokból, vagy akár újabb webszolgáltatásokat is nyújthat belőle. A webszolgáltatások nincsenek egyetlen programozási nyelvre korlátozva, ezért könnyen használhatóak platform-független információ cserére. A nyilvános webszolgáltatások ugyanakkor folyamatos támadási felületet képeznek (pl.: adat injekció[4], hibás adattovábbítás, DoS[6]). Ez az egyik fő ok, hogy miért játszik kulcsfontosságú szerepet a validáció mind a kliens mind a szolgáltató oldalán. A szolgáltatónak biztosítania kell, hogy a kért adat helyes formátumba kerüljön továbbításra, valamint, hogy ne veszélyeztesse a rendszer integritást. A kliens oldalon az eredmények validálása, valamint a rendszer integrációja elsődleges szempont. Jelenleg az egyetlen módja annak, ha bármelyik oldalt validálni kívánjuk, hogy komoly változtatásokat eszközöljünk a rendszerekben. Habár ez nem lehetetlen, mégis jelentős erőforrás igénye van a validációs logika egy meglévő rendszerbe történő integrálásának. Amennyiben pedig változnak az adat

formátum követelmények (pl.: új bankszámla formátum), akkor a szolgáltató rendszert is frissíteni kell, vagy bizonyos esetekben újrafordítani. Hasonló a helyzet a kliens oldalon is, amennyiben a bejövő adatoknak újabb szűrésére van szükségük.

Mivel a webszolgáltatások képesek XML formátumban kommunikálni a klienseikkel (SOAP[3] üzenetekkel), ezért kiváló választás volt az SRML alkalmazásával történő validációra. A jSRML és ProtoML nyelvek kifejlesztése során szerzett tapasztalatokat felhasználva egyesítettük azokat az SRML nyelv elemeivel. Ennek az egyesítésnek az eredménye az SRML 3.0[11] nyelv lett. A jSRML és ProtoML metanyelv segítségével az új SRML nyelv könnyebben használható, precízebben képes szabályokat leírni, amelyek mérete is lényegesen kisebb. Az új metanyelv és a validációs motor (`wsSRML`) a webszolgáltatások területének a validálására is alkalmas. Az új SRML 3.0 formátumban szétválasztottuk az elvárt értékeket a validációs és formai követelményektől. A másik jelentős változás, hogy a validációs motor most már támogatja a függvények egymásba ágyazását (mint ahogyan a ProtoML). Ennek köszönhetően könnyebbé vált a szabályok definíciója.

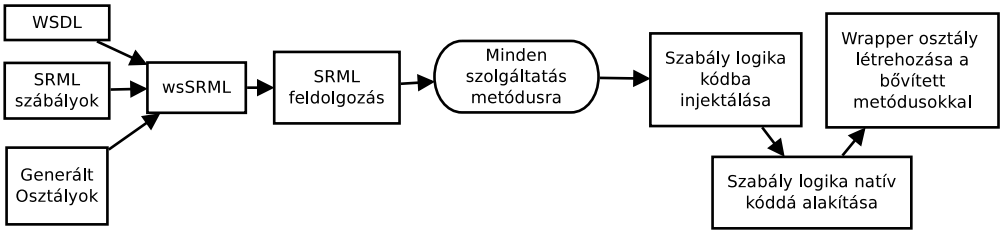
Az SRML 3.0 alapvető szintaktikai és használhatósági újításait az alábbi pontok összegzik:

- Új, függvényorientált szintakszis, amelyben a függvények egymásba ágyazhatók és együtt kiértékelhetők.
- A `conditions` elem lehetőséget biztosít a feltételek listázására. A vizsgált elemeknek eleget kell tenniük ezen feltételeknek. A *match* parameter segítségével vezérelhető, hogy hány feltételnek kell teljesülnie a sikeres validációhoz (*bármelynek*, vagy *mindnek*).
- A `values` elem segítségével kontextus függő érték definíciókat tudunk megadni, amelyeket a motor felülről lefele értékel ki. Az elvárt értékek közül a motor azt választja, amelynek a feltételei legelőször teljesülnek. Ennek segítségével könnyebb az értékek javítása.
- A `validation-record` segítségével megadhatjuk, hogy az XML dokumentumban melyek a rekordok és melyek a fő azonosító attribútumok.
- A `validation-doc-root` elem képes az XML dokumentum gyökér elemét definiálni.

Az új `wsSRML` motor két üzemmódban képes a webszolgáltatásokat validálni. Az első mód a kliens-oldali, ahol a validációs folyamat a generált kódba integrálódik. A másik mód esetén a motor elfogja a célzott webszolgáltatás bejövő és kimenő kommunikációját és a validációs logikát egy proxy szolgáltatásként végzi.

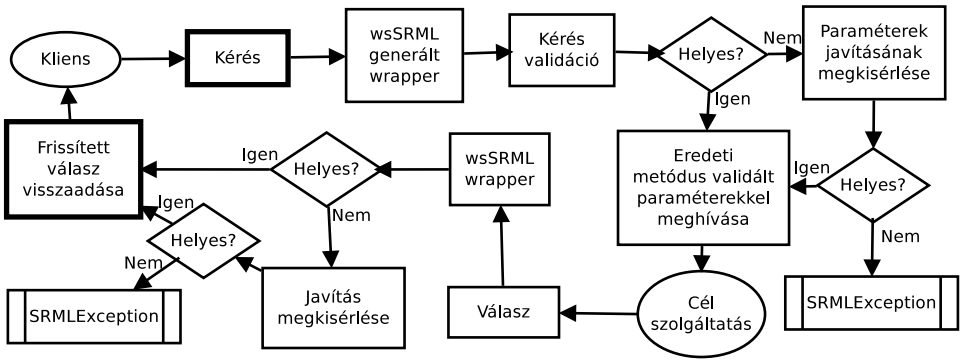
A validációs fázis során mind a *Request* (kérés) és *Response* (válasz) paraméterek elérhetők, mivel a wrapper osztály egy metódusa végzi a tényleges hívást. Ez még kiforrottabb validációs szabályok létrehozását teszi lehetővé, hiszen vannak olyan esetek, ahol a válasz tartalmának részei függnek a kérés paramétereitől. Biztonság és magasabb szintű helyesség érhető el azáltal, hogy mind a válasz és a kérés oldalt is vizsgálja a motor. A legtöbb validációs motor csak a kérés oldalra koncentrál. Lehetnek azonban olyan esetek, amikor egy illetéktelen külső forrás elfogja az adatcsomagot és azt megváltoztatja, hogy a rendszerhez hozzáférhessen. Olyan esetekben is szükség van a válasz validációjára, amikor maga a szolgáltatás is külső webszolgáltatások eredményét továbbítja, melyek közül nem mindegyik helyes. A webszolgáltatás válasza lehet, hogy struktúrálisan helyes, de tartalmi validációját csak bonyolult műveletek során tudjuk garantálni. Az SRML 3.0 segítségével lehetőség van az elvárt értékek definiálására is, amely segítségével javítható a webszolgáltatás kérése és valasza.

Mint azt korábban említettük, a `wsSRML` motort két üzemmódban lehet futtatni: *natív* és *proxyzott*. Mindkettőnek vannak előnyei és hátrányai. A *natív* mód esetén a `wsdl2java` először legenerálja a webszolgáltatás wrapper osztályait és interface-ét. Ezt követően analizálja az SRML szabályokat és kibővíti az osztály kódját (hasonlóan a ProtoML-hez). A *9. ábra* demonstrálja a *natív* mód wrapper osztály generálásának menetét.



9. ábra. Natív mód wrapper osztály generálási folyamata

A motor wrapper osztályt kibővíti az SRML file szabályaiból kinyert üzleti logikával, amelyeket *reflection* segítségével futtat. Mivel a szabályok tartalmazhatnak függvényeket, amelyeket egymásba ágyazhatók, ezért a validációs motor könyvtárát be kell illeszteni abba a projektbe, ahol használni kívánjuk. A *10. ábra* bemutatja hogyan működik a *natív* validációs üzemmód.

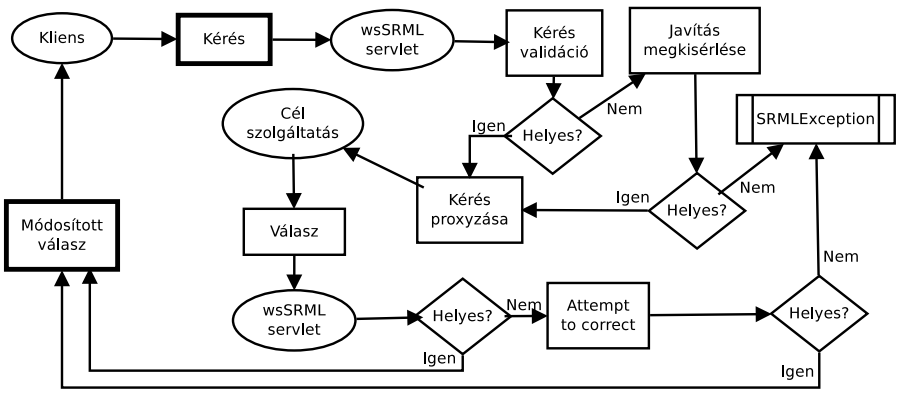


10. ábra. wsSRML natív validációs módja

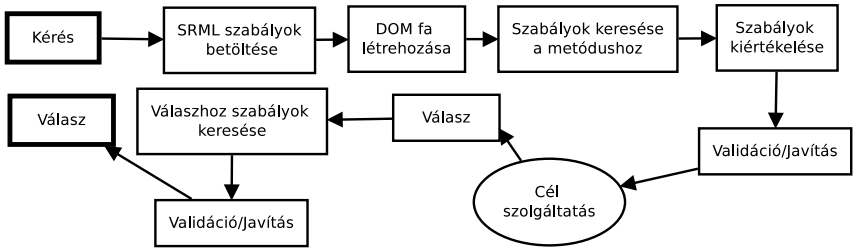
A *natív* mód egyik előnye, hogy gyors fordítást és futást tesz lehetővé, amelyet bármely projektben alkalmazhatunk. A hátránya, hogy nem lehet a rendszer leállítása nélkül lecserélni a validációs szabályokat.

A `wsSRML` második üzemmódja a *proxyzott* mód. Ez a mód olyan esetekben hasznos, ahol a klienst vagy a szervert nem lehet bővíteni a validációs kóddal. A *proxy* megközelítés esetén egy köztes servletet rakunk be a kliens és a szerver közé. A kliens ezen severletnek küldi a webszolgáltatás kérését, amely továbbítja a cél webszolgáltatásnak (hasonlóan a jSRML tanuló moduljánál). A folyamat során a proxy felhasználja az SRML szabályokat, hogy validálja és potenciálisan kijavítsa a bejövő és kimenő forgalmat. A *11. ábra* vázolja a proxyzott validációs folyamatot.

A proxy servletnek három almódja van: *valós-idejű* (real-time rule loading), *lefordított plugin* (compiled rule plugin) és *SOAP*. Mindhárom esetben a request fázisban fogja végezni a validációt és a proxyzást. Ha sikertelen a validáció, akkor dob egy kivételt és a hiba visszaáramlik a hívó félhez. Amennyiben a válasz, amit a szerver adott vissza hibás, a motor megpróbálja kijavítani a szabályok alapján. Ez gyengébb teljesítményű, mint a *natív* üzemmód, viszont nagyobb flexibilitást ad a szabályok cserélésében az érintett rendszerek kikapcsolása nélkül.



11. ábra. Proxyzott validációs folyamat



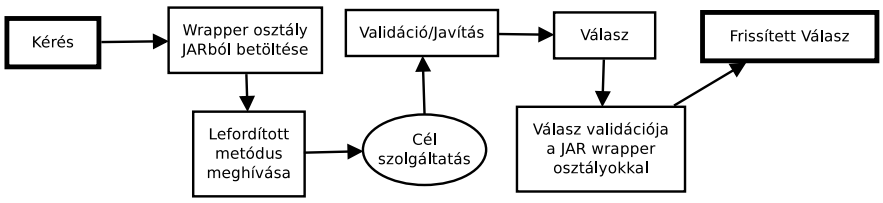
12. ábra. Valós-idejű proxyzott validációs folyamat

A valós-idejű proxy esetén hasonló a kiindulópont, mivel itt is legenerálódik a kibővített wrapper osztály. Ebben az esetben viszont nem fordítja bele a szabályokat, hanem minden híváskor átalakítja a rendszer a paramétereket egy DOM fává a `wsSRML.convertToDOM()` hívásával és végrehajtja rajta a megfelelő szabályokat. A *12. ábra* mutatja a valós-idejű proxy validációs folyamatát.

A harmadik, *„lefordított plugin mód"* esetén a rendszer lefordítja és becsomagolja a szabályokat egy JAR állományba, hasonlóan a natív üzemmódhoz. Az előnye ennek az, hogy előre legeneráljuk a validációs logikát, amelynek alapján nem kell minden kérés esetén feldolgozni a szabályokat. Ez a mód jelentősen gyorsabb, mint a *valós-idejű* feldolgozás, mivel ilyenkor nem kell a DOM fát paraméterekből felépíteni hívásonként. A hátránya, hogy nehezebb az üzleti logikát lecserélni. A *13. ábra* mutatja a lefordított plugin üzemmódot. Minden webszolgáltatásnak, amit ebben az üzemmódban használnak, saját kontextusa és classloadere lesz. Itt egy nagy kihívás az, hogy a Java nem enged több verziót használni ugyanabból az osztályból. Ennek a megoldására egy OSGi-hoz hasonló megközelítést alkalmaztunk. A pluginok és összes osztályuk saját „homokozó" környezetbe kerülnek.

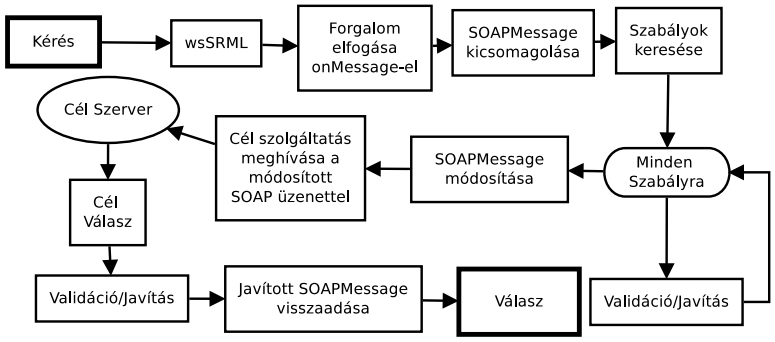
A `wsSRML` proxy servlet indulásakor betölti az összes plugin JAR állományát, majd mindegyiket saját végpontként expozálja. Ennek segítségével egy wsSRML servlet számos más webszolgáltatás proxy kiszolgálójaként működhet. Ez lehetőséget ad egy *validáció-mint-szolgáltatás* (VaaS)- modellnek is.

A `wsSRML` harmadik módja a nyers SOAP üzenetek elfogására szakosodott. Ilyenkor teljes proxyról beszélünk, hiszen nincs wrapper osztály generálás és kód augmentálás. Ebben az üzemmódban a proxy servlet fogadja a bejövő SOAP üzenetet és arra alkalmazza az SRML szabályokat. Ez a megközelítés hasonló a *valós-idejű* üzemmódhoz, mivel a szabályokat minden SOAP üzenet esetén megkeresi a rendszer és alkalmazza őket. A megoldás sokkal átláthatóbb, mivel nincs felesleges fordítás és wrapper osztály se. A SOAP üzenetet DOM-fává alakítja (mivel a SOAP is XML alapú) és azon hajtja végre a



13. ábra. Lefordított Plugin proxyzott validációs folyamat

szabályokat. Ennek az üzemmódnak a sebessége szuboptimális, mivel a szabályokat minden kérés és válasz esetén be kell tölteni és feldolgozni. A *14 .ábra* mutatja a SOAP üzemmódját a motornak.



14. ábra. SOAP proxyzott validációs folyamat

A proxy mód előnye, hogy olyan esetben is használható, amikor a kliens és/vagy szerver forráskódja nem érhető el, vagy nem módosítható. Lehetőséget biztosít a validációs szabályok *valós-idejű* cseréjére potenciális kiesés nélkül. A hátránya az, hogy jelentősen lassabb a többi üzemmódhoz képest és működéséhez szükséges egy proxy servlet is, amely növelheti a rendszer komplexitását.

4.1. Tézis összefoglalása és saját eredmények

- Az SRML 2.0, jSRML, ProtoML nyelvek integrációjával létrejött az SRML 3.0 metanyelv. Ez a nyelv a többi nyelv összes előnyét átvette, amely eredményeként egy új, hatékonyabb validációs nyelv jött létre.

- Az új 3.0 kiterjesztés függvény-orientált szabály definicókon alapul, ahol a függvényeket egymásba ágyazhatjuk (hasonlóan a ProtoML-hez). Ennek köszönhetően sokkal átláthatóbbak a szabályok, jelentősen rövidebbek és gyorsabban feldolgozhatók.

- A kiterjesztés szétválasztja a feltételekre vonatkozó szabályokat az elvárt értékekre irányuló szabályoktól, mely eredményeképp jobban áttekinthetőek lesznek a definíciók.

- A `wsSRML` validációs motor képes validálni és kijavítani a webszolgáltatások *Kérését* (Request) és *Válaszát* (Response). A motor két üzemmódban alkalmazható: *natív* és *proxyzott*.

- A motor, felhasználva az SRML 3.0 szabályokat képes Java kódot generálni a szabályokból és ezeket beleinjektálni a `wsdl2java` által generált wrapper osztályokba. Ennek segítségével a validációs logikát a tényleges szolgáltatás hívásával együtt végezhetjük. A motor, proxyzott üzemmód esetén elfogja a szolgáltatás forgalmát egy köztes servlet segítségével. A szolgáltatás

ezt követően validálja illetve kijavítja az üzeneteket, mielőtt továbbítaná az eredeti webszolgáltatásnak. Ennek az üzemmódnak van egy *plugin* almódja is, amelynek köszönhetően a servlet képes számos webszolgáltatás validációját ellátni (hasonlóan, ahogy a jSRMLTool VaaS megközelítése működött).

Az SRML kiterjesztése webszolgáltatások validálására teljes mértékben az én tudományos munkám eredménye, melyet a [11] publikáció tartalmaz.

5. Összefoglaló

A disszertáció szemlélteti, hogy miként került az SRML nyelv kibővítésre, annak érdekében hogy alkalmazható legyen a validáció terén is. A nyelv fejlődése során számos metanyelv jött létre, amely jelentősen hozzájárult a végső SRML 3.0 kialakításában. A dolgozat bemutatta, milyen módszerrel lehet XML dokumentumokat, webes űrlapokat, Google Protocol Buffers üzeneteket és webszolgáltatásokat validálni. Ezek a területek lefedik a rendszerek közti információcsere leggyakoribb módjait, mely alapján releváns témát jelentenek. A jövőben tovább akarjuk terjeszteni a nyelv korlátait a bináris állományok validációjára, illetve validációs eljárást nyújtani az klasztereken osztott dokumentumok validációjára is (pl.: Hadoop).

Hivatkozások

[1] Document object model (DOM), http://www.w3.org/dom/.

[2] H2 database engine, http://www.h2database.com/html/main.html.

[3] D. Box and D. Ehnebuske. Simple Object Access Protocol (SOAP) 1.1. Technical report, World Wide Web Consortium, http://www.w3.org/TR/SOAP/, 2000.

[4] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL injection attacks. In LNCS, editor, *International Conference on Applied Cryptography and Network Security (ACNS)*, volume 2, 2004.

[5] Google. Protocol Buffer, http://code.google.com/apis/protocolbuffers/docs/overview.html, 2008.

[6] M. Handley. Internet Denial-of-Service Considerations. Technical report, IAB, RFC4732, 2006.

[7] J. Hunter. JDOM, http://jdom.org/docs/apidocs/, 2000.

[8] J. Hunter and W. Crawford. *Java Servlet Programming*. O'Reilly, 2nd edition, 2001.

[9] M. Kálmán. ProtoML: A rule-based validation language for Google Protocol Buffers. In *Proceedings of the 8th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 193–198, London, UK, December 9-12 2013, IEEE Computer Society.

[10] M. Kálmán. Versatile form validation using jSRML. *Acta Cybernetica*, 2014 (Accepted for publication).

[11] M. Kalman. Rule-based web service validation. In *Proceedings of the 21st International Conference on Web Services (ICWS)*, Alaska, USA, June 27 - July 2 2014 (Accepted for publication), IEEE Computer Society.

[12] M. Kálmán and F. Havasi. Enhanced XML validation using SRML. *International Journal of Web & Semantic Technology (IJWeST)*, Volume 4(October):1–18, 2013.

[13] M. Kálmán, F. Havasi, and T. Gyimóthy. Compacting XML documents. In *Journal of Information and Software Technology*, volume 48, pages 90–106. Elsevier, February 2006.

[14] D. Raggett and A.L Hors. HTML 4.0 specification. Technical report, W3C, April 1998.

[15] J.E. Refsnes. Introduction to DTD, http://www.w3schools.com/dtd/dtd_intro.asp.

[16] E. van der Vlist. *XML Schema*. O'Reilly, 2001.

Nyilatkozatok

Érintett cikk: [12]

Társszerzői nyilatkozat

Kijelentem, hogy ismerem Kálmán Miklós doktorjelölt *„Web-alapú metanyelvek dokumentumainak validációja szemantikus szabályokkal”* című disszertációját.

A disszertációban szereplő közös eredményekre vonatkozóan kijelentem, hogy a következő eredményekben a pályázó hozzájárulása volt a meghatározó:

- SRML XSD-re való adoptálása
- XML validálása szemantikus szabályokkal (SRML 2.0)
- Az SRML nyelv kiterjesztése (1.1, 2.0, 3.0 verziókra), amely lehetővé teszi validációra való alkalmazását
- Az XML validáció során a dokumentum javításának mechanizmusa

A következő eredményekhez való hozzájárulásunk oszthatatlan:

- SRML használata XML tömörítés hatékonyabbá tételére
- SRML 1.0 gépi tanulása
- SRML kézi definiálásának kombinációja gépi tanulással

A következő eredményekben az én hozzájárulásom volt meghatározó:

- XML szemantikus kiterjesztés (SRML) elvének kidolgozása
- XML kompakció elvének kidolgozása
- Kézzel írt SRML szabályokkal történő CPPML kompakció

Havasi Ferenc

Havasi Ferenc

Szeged, 2014. május. 20.

Témavezetői nyilatkozat

Kijelentem, hogy ismerem Kálmán Miklós doktorjelölt *„Web-alapú metanyelvek dokumentumainak validációja szemantikus szabályokkal”* című disszertációját, a fenti publikáció alább felsorolt eredményeiben a Jelölt hozzájárulása meghatározó volt: □

1. Az SRML nyelv kiterjesztése validációra (SRML 2.0).
2. Az SRML szablyok integrálása az XSD dokumentumba.
3. Adatbázisok validációja SRML szabályokkal.
4. Webes űrlapok validációja szemantikus szabályokkal és a jSRML nyelv kifejlesztése.
5. Google Protocol Buffers üzenetek validációja és a ProtoML nyelv kifejlesztése.
6. Az SRML nyelv kiterjesztése a webszolgáltatások terére illetve az SRML 3.0 nyelv kidolgozása.

.....□

Prof. Gyimóthy Tibor
temavezető

2014. május 19.