INTELLIGENS WEBRENDSZEREK

Egyetemi doktori (PhD) értekezés

NAGY ZSOLT

TÉMAVEZETŐ: DR. BODA ISTVÁN

DEBRECENI EGYETEM

Természettudományi Doktori Tanács

Informatikai Tudományok Doktori Iskola

Debrecen, 2014

Ezen értekezést a Debreceni Egyetem Természettudományi Doktori Tanács Informatika Tudományok Doktori Iskola Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és alkalmazásai programja keretében készítettem a Debreceni Egyetem természettudományi doktori (PhD) fokozatának elnyerése céljából.

Debrecen, 2014. ….....

Nagy Zsolt
doktorjelölt

Tanúsítom, hogy Nagy Zsolt doktorjelölt 2008 – 2014 között a fent megnevezett Doktori Iskola Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és alkalmazásai programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Az értekezés elfogadását javasolom.

Debrecen, 2014. ….....

Dr. Boda István
témavezető

INTELLIGENS WEBRENDSZEREK

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
az informatika tudományágban

Írta: Nagy Zsolt okleveles programtervező matematikus

Készült a Debreceni Egyetem Informatika Tudományok Doktori Iskolája
(Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és alkalmazásai programja) keretében

Témavezető: Dr. Boda István

A doktori szigorlati bizottság:

elnök:	Dr. Terdik György
tagok:	Dr. Bognár Katalin
	Dr. Kovács László

A doktori szigorlat időpontja: 2013. május 30.

Az értekezés bírálói:

Dr. ....
Dr. ....

A bírálóbizottság:

elnök:	Dr.	.....
tagok:	Dr.	.....
	Dr.	.....
	Dr.	.....
	Dr.	.....

Az értekezés védésének időpontja: 20…… … .

Köszönetnyilvánítás

Köszönettel tartozom kollégáimnak, oktatóimnak, hogy a doktori tanulmányaim során mindvégig támogattak és hasznos tanácsaikkal, javaslataikkal segítették értekezésem létrejöttét.

Külön köszönöm témavezetőm, Dr. Boda István áldozatos munkáját, aki atyai gondossággal figyelte, irányította kutatásaimat, tudományos barangolásaimat.

Hálával tartozom a Debreceni Egyetem Informatikai Tudományok Doktori Iskolájának, amiért lehetővé tették, hogy a számomra oly kedves alma materembe végezhettem PhD tanulmányaimat.

Köszönöm értekezésem bírálóinak fáradhatatlan munkáját, akik szabadidejüket feláldozva azon dolgoznak, hogy hasznos tanácsaikkal még jobbá tegyék jelenlegi és leendő kutatási eredményeimet, segítsék kutatói pályámat.

Végezetül hálásan köszönöm barátaim, családtagjaim, feleségem támogatását és türelmét azért a sok ellopott időért, amit az olykor éjszakákba nyúló kutatói lelkesedésem vett el tőlük.

Tartalomjegyzék

Bevezetés

A disszertáció 6, egymáshoz szorosan kapcsolódó fejezetet tartalmaz, melyekben a web alapú rendszerfejlesztés teljes folyamatát végigkísérem. A fejezetekről készült rövid áttekintésekben ismertetem a kiindulási problémát, a problémára adott új kutatási eredményeimet, módszereimet.

Fiatal kutatóként még épp, hogy csak betekintést kaptam a tudományos világba, így külön öröm számomra, hogy az értekezésben megfogalmazott eredményeimet, cikkeimet külföldi kutatók idézik, használják, így kérem engedtessék meg, hogy ezeket is feltüntessem, megemlítsem ott, ahol ez releváns.

Az *első fejezet*ben a webes rendszerek fejlesztési folyamatát tekintettem át, a jelenleg használt modell hiányosságait felderítve a mai kor igényeinek megfelelő megoldást, új módszert dolgoztam ki.

A *második fejezet* a teljes fejlesztési folyamatból kiemeli a rendszertervezést, részletesen ismerteti a manapság alkalmazott hagyományos és divatos tervezési mintákat, majd az eddigiekhez képest új, ugyanakkor mégis MVC alapokon nyugvó tervezési mintát dolgoztam ki, mely egyben alkalmas a kliens-szerver keretrendszerek integrálására is.

A *harmadik fejezet* ismerteti a jelenleg piacvezető webrendszerek teljesítménybeli hiányosságait, a meglévő sebességnövelő technológiákat sorra veszi, ellenőrzi, majd egy olyan új folyamatmodellt kínál, mellyel gyors webrendszerek építhetők, a meglévő webrendszerek sebessége növelhető.

A *negyedik fejezet* egy olyan érdekes, Ajax-alapú, implicit adatgyűjtő technológiát mutat be, mellyel igen hatékonnyá tehető az információgyűjtés, kiszolgálja a reszponzív megjelenítés igényeit, valamint ideális adatgyűjtési lehetőséget kínál az ajánlói rendszerek számára.

Az *ötödik fejezet* a szoftverrendszer felhasználóval közvetlenül kapcsolatban lévő komponensét hivatott intelligenssé, jobbá tenni. Az itt kínált új modellel megvalósuló fejlesztések felhasználói interfésze intelligens módon alkalmazkodik az őt megjelenítő eszközhöz, annak típusához, képernyőfelbontásához.

Végezetül az utolsó, *hatodik fejezet*ben felkutatom, elemzem és bemutatom a már kész intelligens rendszerek adatvédelmi kockázatait, rávilágítok arra, milyen veszélyeknek vannak kitéve személyes adataink a Web 2.0 korában.

Új eredmények

Az első fejezet áttekintése: Adaptív fejlesztési módszer

Probléma

A szoftverfejlesztési folyamat bonyolult és összetett, az egyes részfeladatokat, illetve az ezek közötti kapcsolatokat modellezni szükséges annak érdekében, hogy egyrészt áttekinthető, másrészt moduláris felépítésű legyen. A legismertebb és legelterjedtebb a vízesés (waterfall) modell. A modell jól alkalmazható abban az esetben, ha a rendszerrel szemben támasztott követelményeket már a fejlesztés legelején tudjuk.

Sajnos a való életben a megrendelő nem tudja teljes pontossággal definiálni a projekt elején, mit is akar; a követelmények menet közben változnak, finomodnak. Épp ezért a vízesés modellt az üzleti webfejlesztésben egyre kevésbé használják, ugyanis ezekre a változásokra ez a modell nincs felkészülve; ha valamely fejlesztési fázis lezárult, szinte lehetetlen azon változtatásokat eszközölni.

Megoldás

Egy olyan módszerre van szükség, mely rugalmas, lehetővé teszi azt, hogy menet közben rendszeresen konzultáljunk a megrendelővel, és ha módosításra van szükség, azt a legkisebb költséggel tegye lehetővé. Ilyen módszernek ígérkezik az agilis szoftverfejlesztés, ám sok esetben az sem megfelelő. A két rendszert a saját kutatás-fejlesztési munkám eredményei alapján ötvöztem és ebből alkottam meg a mai modern webrendszerek fejlesztéséhez igazodó új folyamatmodellt, az adaptív fejlesztési módszert.

A friss kutatási eredményeket *Adaptive Design Process for Responsive Web Development* (DOI: 10.13140/2.1.3354.5601) című cikkemben is ismertettem.

A második fejezet áttekintése: Integrált tervezési minta

Probléma

A mai webalkalmazások fejlesztése megfelelő tervezési minta nélkül nem lehetséges, hisz ma már egyszerre kell ellátni a gazdag kliens oldali programozási feladatokat a megszokott szerver oldali alkalmazásfejlesztéssel. Akár a kliens oldali akár a szerver oldali programozási munkáról beszélünk, a munka volumene megköveteli a tervezési minták használatát. Halmozottan igaz ez egy komplex webalkalmazásra, ahol a kliens-szerver oldali fejlesztés szükségszerűen elválaszthatatlan kapcsolatban áll egymással. A legnépszerűbb tervezési minta mind a mai napig az MVC. A kutatás ezen fázisában arra kerestem a választ, hogy a mai modern fejlesztői környezetekkel is ugyanúgy használható e a tervezési minta, tudok e esetleg egy új, jobb, a kliens és szerver oldali rendszereket összefogni képes architektúrát kínálni a fejlesztőknek.

Megoldás

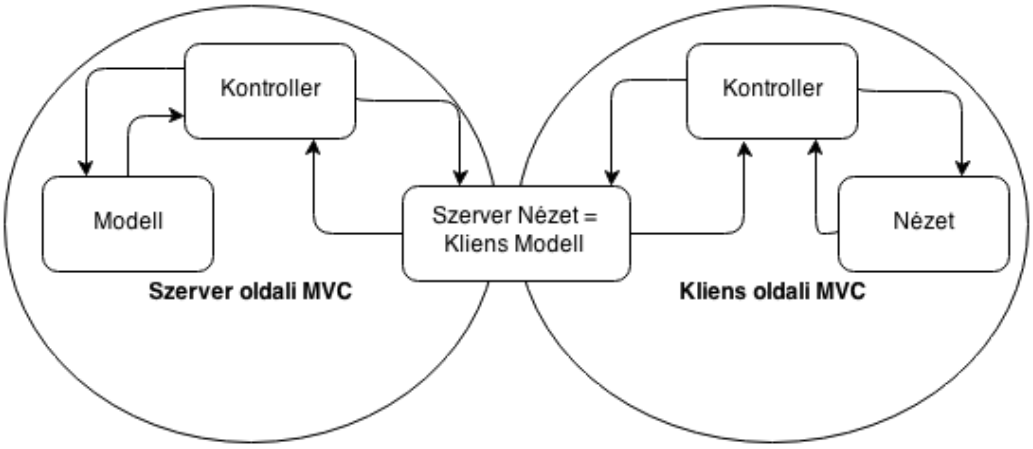
A különböző tervezési minták vizsgálata során arra a következtetésre jutottam, hogy az eredeti MVC architektúra kissé módosított, Cocoa verziója a megfelelő kiinduló állapot egy új integrált minta kidolgozására. Választásomat indokolta továbbá az a tény, hogy mind kliens, mind szerver oldalon az MVC keretrendszerek a legelterjedtebbek.

A kérdés továbbá az volt, hogyan lehet összekapcsolni a népszerű JavaScript MVC-ket a szintén igen elterjedt PHP MVC-kel úgy, hogy közben az összekapcsolt rendszerünk ugyanúgy megfeleljen a Modell-Nézet-Kontroller felépítésnek.

Szerver oldalról megközelítve egyértelműen a Nézet komponens az, ami további szegmentálásra szorul, hisz ennek bonyolultsága az, ami nehezíti a fejlesztési munkát. Lecserélve a Nézetet egy komplett kliens oldali MVC-re a rendszerünk MVC-ből M(MVC)C-é változik.

Amennyiben kliens oldalról vizsgáljuk a helyzetet, az MVC architektúra Modell komponense a sima HTML kód, a Nézet a CSS fájl (ahány CSS fájl, annyi nézet), míg a Kontroller maga a böngésző, illetve a böngésző képességeit kiterjesztő JavaScript programkód. Egy másfajta megközelítésben – különösen ha a kliens-szerver között asszinkron, például AJAX-alapú kommunikáció zajlik, a Nézet a HTML+CSS+adat kombinációjából megszületett felhasználói felület, a Kontroller szerepét JavaScript osztályok és metódusok töltik be, míg a Modell nem más, mint a webszervertől érkező adat.

Bármelyik szemléletet is tekintjük, kliens oldali megközelítésben a Modell az a komponens, melyen keresztül a rendszerünk illeszthető a szerveri oldali MVC rendszerhez. Ennek alapján az alábbi sematikus ábra szemlélteti az új tervezési mintát.



1. ábra: Integrált tervezési minta

A kidolgozott új mintát a SOFA2014 nemzetközi konferencián ismertettem, melyből már megjelentetésre elfogadott *Integrated Design Pattern for Intelligent Web Applications* címmel született 14 oldalas cikkem.

A harmadik fejezet áttekintése: Modell a webrendszerek gyorsabbá tételére

Probléma

Ahogy a bevezetőben is ismertettem, megannyi kutatás igazolja: mit sem ér egy intelligens, remek matematikai modellre épülő webrendszer, ha a felhasználó nem győzi kivárni a megjelenítendő tartalmat. A sebesség fontosságának ékes bizonyítéka, hogy 2010-től a Google is felvette és alkalmazza a weboldalak sebességét, - mint értékelési paramétert - a PageRank rangsorolási algoritmusában. Több érdekes könyv és tudományos cikk jelent meg a weblapok gyorsabbá tételére, sőt a két legnagyobb kereső, a Google és a

Yahoo is közzétett számos ajánlást és mérési módszert ezzel kapcsolatban, ám felmérések alapján a világon kevés olyan weboldal van, mely maradéktalanul megfelelne a javasolt technológiai irányelveknek. A világ, az Egyesült Államok, illetve Magyarország első 10 leglátogatottabb weboldalát megvizsgálva kiderül, hogy hazánk igen csak le van maradva ezen új javaslatok és módszerek alkalmazásában.

Megoldás

Kutatásom harmadik fázisában egy napi 4000 egyedi látogatót számláló turisztikai portálon teszteltem és alkalmaztam a saját fejlesztési tapasztalatokon alapuló, illetve a szakirodalom által javasolt módszereket. A hatékonyság-vizsgálat eredményeire építve egy olyan módszergyűjteményt alkottam meg, mely alkalmas nem csak meglévő weblapok átalakítására, hanem a tervezési fázisban is jól alkalmazható, így nagymértékben elősegítheti a gyorsabb intelligens webrendszerek születését.

Improved Speed on Intelligent Web Sites címmel 2013-ban közöltem ide vonatkozó eredményeimet, melyre **egyiptomi** kutatók is felfigyeltek, az International Journal of Advanced Computer Science and Applications folyóiratban megjelent *XML Schema-Based Minification for Communication of Security Information and Event Management (SIEM) Systems in Cloud Environments* (DOI: 0.14569/IJACSA.2014.050912) cikkükben hivatkoznak munkámra.

A negyedik fejezet áttekintése: Implicit adatgyűjtés, egy új modell

Probléma

A személyre szabott tartalom megvalósításának legfőbb eszköze az ajánlói rendszerek alkalmazása, használata. Azonban a jelenlegi kutatások java azzal a problémával szembesül, hogy igen nehéz megfelelő mennyiségű és minőségű valós adatot összegyűjteni, ezért a legtöbben vagy minta-adatbázisokon dolgoznak vagy – jóval kevesebb számban - olyan szerencsés helyzetben vannak, hogy valódi rendszereken tesztelhetik a különböző adat- vagy szövegbányászati módszereiket.

Ám még az olyan nagy látogatottságú és ismert webrendszereknél, mint az amazon.com vagy az eBay.com is igen nehéz a felhasználókat rávenni arra, hogy minduntalan értékeléseket, preferencia értékeket adjanak meg; egyrészt

időigényes feladat, másrészt a felhasználók bizalmatlanok, egyre kevésbé adnak ki bármilyen információt is magukról.

Megoldás

Épp ezért egyre inkább az implicit adatgyűjtés felé mozdulnak az ez irányú kutatások. Az implicit adatgyűjtés során a web alapú rendszerek a felhasználó tudta, beavatkozása nélkül képesek folyamatosan, igen nagy mennyiségű adatot gyűjteni, majd azokat feldolgozni. A kutatásom ezen részében egy új, AJAX alapú technológiát és modellt kínálok az adatgyűjtés hatékonyabbá tételére.

Bár az eredményeket még 2012-ben közöltem az *AJAX-Based Data Collection Method for Recommender Systems* című cikkemben, az ott ismertetett módszer még ma is aktuális, idén egy **koreai** szerző trió, Sung Moon Bae és társai idézték a *Utilization of Demographic Analysis with IMDB User Ratings on the Recommendation of Movies* (DOI: 10.7838/jsebs.2014.19.3.125) cikkükben.

Az ötödik fejezet áttekintése: Reszponzív és tartalomfüggő megjelenítés

Probléma

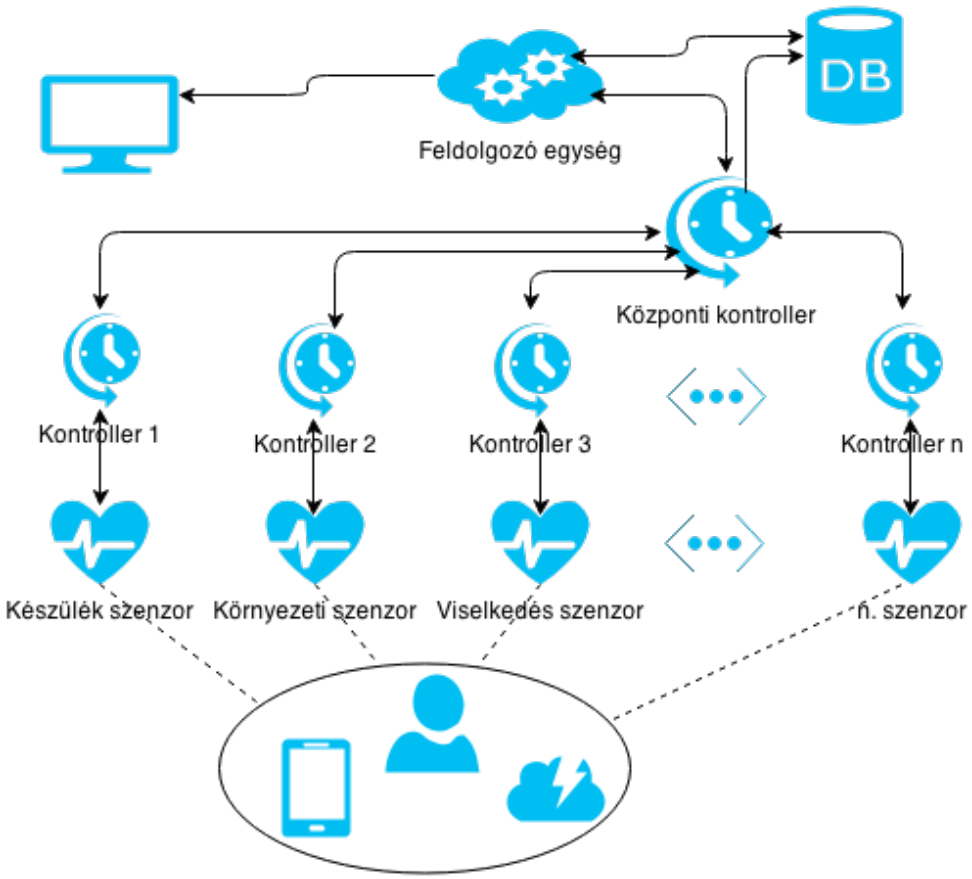
A személyre szabott tartalom rendkívül fontos, az még inkább, hogy azt a felhasználónak tetsző módon szolgáltassák a rendszerek. A korai webrendszerek egyik legnagyobb hátránya, hogy a felhasználónak nem biztosították az ún. "alkalmazás érzetet", a desktop alkalmazások során megszokott "kattintunk és már azonnal ott is a tartalom" a web kliens-szerver architektúrájának sajátosságából adódóan nem érvényesült; amíg a böngésző a kérésére kapott választ nem kapja vissza a szervertől, addig vár, így a felhasználó is. Mindannyian töltöttünk már súlyos másodperceket úgy a számítógép előtt, hogy vártuk, egy weblap betöltődjön, frissüljön. Ezt a hiányosságot a fejlesztők az AJAX asszinkron kommunikációs képességeivel igyekeznek orvosolni.

Mindezek mellett a mobil eszközök térhódításának köszönhetően, valamint amiatt, hogy az IPv6 bevezetésével akár a hűtőszekrényünkről vagy a kávéfőzőnkről is csatlakozhatunk az Internetre, egy új kihívással kell szembenézniük a fejlesztőknek: már nem elég tartalomfüggő webrendszereket tervezni, megvalósítani, az eszközfüggő reszponzív weboldalaké a jövő.

Megoldás

Ennek megfelelően az ismertetett modellel megvalósuló fejlesztések felhasználói interfésze intelligens módon alkalmazkodik az őt megjelenítő eszközhöz, annak típusához, képernyőfelbontásához. A felhasználóról szerzett információkat főbb jellemzőik alapján 3 különböző csoportba osztottuk, az egyes csoportok a Készülék, a Környezet és a Viselkedés kategóriákat jelentik és a fejezetben ismertetett szenzor-architektúra (2. ábra) szerint végzik az adatgyűjtést.

A begyűjtött információk alapján aztán az alkalmazásszerverünk összeállítja a felhasználó számára legmegfelelőbb tartalmat, felhasználói interfészt és megküldi a kliens eszköz számára.



2. ábra: Szenzor architektúra

A hatodik fejezet áttekintése: Adatvédelem szakértői szemmel

Probléma

Joggal merülnek fel minduntalan személyiségi jogi kérdések az intelligens webrendszerek kapcsán, hisz napjaink egyik kulcskérdése az informatikai biztonság, a személyes adatok védelme. A láthatatlan és folyamatos adatgyűjtés – nem csak az intelligens rendszerek, hanem a teljes Interneten töltött életünk szempontjából – kiemelt jelentőséggel bír.

Igazságügyi szakértőként több olyan büntető ügyben is végeztem szakértői munkát, ahol az Internet használat által hagyott 'lábnyomokat' kellett felkutatni egy adott számítógépen.

Megoldás

A tapasztalatok meglepőek és tanulságosak. Volt szerencsém számos aspektusból vizsgálni a kérdést; úgy a felhasználó, mint a szakértő, a feltételezett bűnelkövető vagy a nyomozóhatóság szemszögéből. Kutatásom utolsó fázisa annak felderítésére és rendszerezésére irányult, hogy a webes rendszerek milyen információkat gyűjtenek, tárolnak a felhasználókról, hogyan, milyen eszközökkel, módszerekkel lehetséges ezeket az információkat felkutatni, illetve védekezni az ellen, hogy illetéktelen kezekbe kerüljön.

A témában született eredményeimről 2012-ben kettő cikket is írtam. Az első *Using Forensic Techniques for Internet Activity Reconstruction* néven, melyet **kínai** kutatók, Chen Long és társai *User browsing-data recovery of Google browser in private-browsing mode* (DOI: 0.3979/j.issn.1673-825X.2013.06.027) című írásukban idéznek, míg a második, a *Social media risks from forensic point of view* 2014-ben találtatott hasznosnak Mohammad Reza Keyvanpour és **iráni** kutatótársai számára a *Digital Forensics 2.0* (DOI: 10.1007/978-3-319-05885-6_2) publikációjukhoz használták fel. Külön öröm, hogy ez utóbbi cikkemet az **amerikai Védelmi Minisztérium** (Department of Defense) által fenntartott Cyber Security & Information Systems Information Analysis Center (CSIAC) weboldala[1] is említi.

1 https://sw.csiac.org/techs/abstract/549148#.VDl_hb6A3dk

Motiváció és irodalmi áttekintés

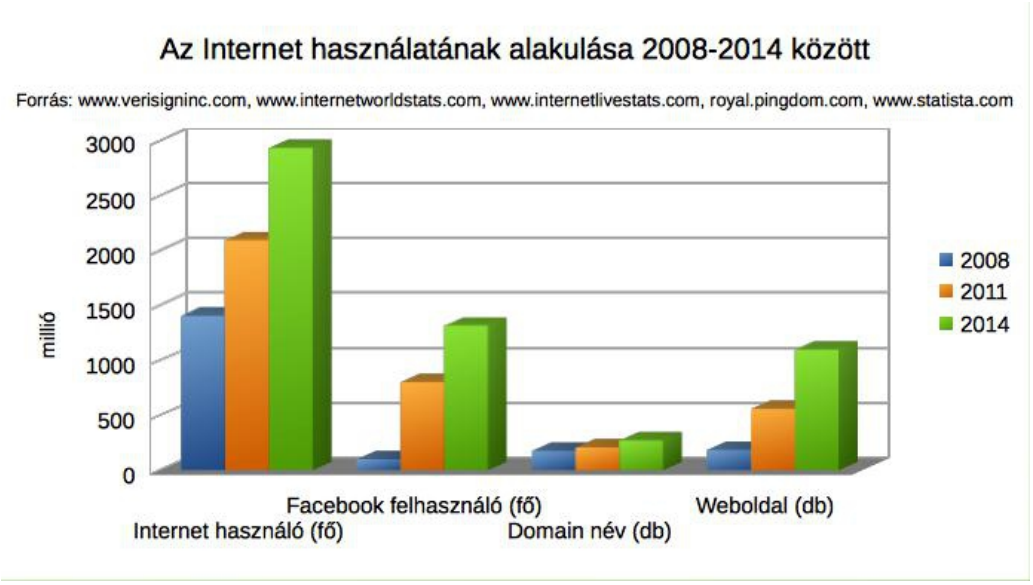
2012-ben, útban egy nemzetközi konferenciára, a repülőgépen kezembe akadt a Traveller magazin aktuális száma, ahol Andrew Hankinson, “Holiday 2.0: are we changing the way we travel forever?” című írásában 4 oldalon keresztül boncolgatja az online foglalási rendszerek és a közösségi oldalak turizmusban betöltött kiemelkedő szerepét. Sorra veszi azokat a webportálokat és szolgáltatásokat, melyek segítségével a komplett nyaralását le tudta szervezni, utazással, szállással, étkezéssel, fakultatív programokkal, baráti találkozókkal.

Elemzi a jelenséget mind újságírói, mind befektetői szemmel, végül több következtetést is levon, mellyel saját tapasztalataink alapján mélyen egyet kell értenünk: minden piaci szereplő számára elkerülhetetlen és a talpon maradás feltétele az Internet adta online (e-) kereskedelmi lehetőségek kiaknázása. Mit sem ér azonban a látványos webportál, a gondos marketingmunkával megfogalmazott szállásajánlat, ha a látogató, az utazni vágyó nem bízik az adott weboldalban. A bizalom a mai Internetes életünk egyik legértékesebb pénzneme, statisztikák igazolják [19] [20], hogy a közösségi portálok, a közösség, és azon belül a közösség véleményét leginkább befolyásoló személyek (véleményformálók, vagy a Barabási-paradigma szóhasználatában: hálózati „középpontok”, „döntésvezetők”, „befolyásos felhasználók”, „befolyásolók”, ld. Barabási [21] 2013:144-136 és 280) véleménye meghatározó szerepet játszik üzleti döntéseinkben, az utazni vágyók jelentős része bízik inkább utastársaik közösségi oldalakon közzétett véleményében, mint például egy utazási iroda által leírtakban.

Egy webportállal, egy informatikai rendszerrel szemben egy másik fontos bizalmi kérdés is felmerül: ez pedig nem más, mint a személyes és pénzügyi adataink védelme. Ebben az esetben már nem egy helyes vagy helytelen üzleti döntés a tét, hanem - olykor – a teljes egzisztenciánk. Így, amikor naponta adatlopásokról, megfigyelésekről, lehallgatásokról hallunk, ne lepődjünk meg azon, ha a felhasználók rendkívül bizalmatlanok. Ez a fajta bizalmatlanság jócskán megnehezíti az intelligens webrendszerek készítőinek a dolgát, és kiemelt feladatot ró a marketing- és informatikai szakemberek számára, hogy megfelelő bizalmat ébresszenek, tudatosítsanak egy-egy márkában, szolgáltatásban, webportálban.

Valóban megváltoztatja életünket és vásárlási szokásainkat az Internet?

Kétségtelen. Az Internetet használók és azon információt kereső felhasználók száma is folyamatosan növekszik. Míg 2008-ban ez a szám 1,4 milliárd volt, 2014-ben már 2,9 milliárd, azaz a Föld lakosságának több mint 40%-a használja rendszeresen az Internet adta lehetőségeket [3. ábra]. Ez annyit tesz, hogy minden egyes másodpercben 7 új, információra éhes felhasználó jelenik meg az Internet világában.



3. ábra: Az Internet használatának alakulása

Kutatásom úgy kezdeti, mint jelenlegi fő célkitűzése a mai web alapú rendszerek hatékonyabbá tétele, mind normál számítógépes, mind mobil platformon. A hatékony webrendszer intelligens; személyre (nem, életkor, foglalkozás, nemzetiség, érdeklődési kör, földrajzi, nyelvi preferenciák) és eszközre (számítógép, telefon, tablet vagy épp hűtőszekrény kijelző) szabott tartalmat szolgáltat a felhasználó számára. Teszi mindezt a lehető leggyorsabban, hisz a kutatások igazolják: mindössze 3 másodperc várakozás és már a látogatók 57%-a ott is hagyta a webes tartalmat. További statisztikák támasztják alá, hogy minden egyes másodperc késedelem dollármilliókat vesz ki a nagyforgalmú webportál tulajdonosok zsebéből [22].

Ennek fényében érthető, hogy miért van kiemelt jelentősége az üzleti világban a hatékony, gyors és a felhasználónak is tetsző webrendszereknek.

Korunk trendjeinek megfelelően egyre inkább az tapasztalható, hogy az Internet lesz éppúgy az elsődleges információ és hírforrás, mint a szolgáltatások és kereskedelmi tevékenységek színhelye. A mai kor embere Internetes oldalakon rendel terméket, szolgáltatást, Interneten olvassa el a napi híreket, sőt a Web 2.0 megjelenése óta blogot, fórumokat, élménybeszámolókat ír, közösségi portálokon keresztül tájékozódik barátai, rokonai felől.

A Web 2.0 fogalom először [1] cikkében jelent meg, ismertséget azonban Tim O'Reilly és Dale Dougherty szerzett a kifejezésnek, a 2005-ben szervezett Web 2.0 konferencián. Mára már a fogalom jóval több, mint marketing kifejezés, valójában új technológiák összefoglaló neve [2].

Noha 2005-ben jelent meg a Web 2.0, Tim Berners-Lee, a web atyja már 1994-ben felhívta a figyelmet arra az igényre, hogy a webet a gépek számára is értelmezhetővé kellene tenni [3], majd 2001-ben [4] a nagyközönség elé is tárta a szemantikus web koncepcióját, melyet már akkor a web következő lépcsőfokaként aposztrofáltak, manapság pedig egyre gyakrabban a web 3.0 néven emlegetik. A furcsa időzavar oka, hogy a szemantikus web koncepció már a Web 2.0 előtt megjelent, ám annak bonyolultsága és összetettsége miatt mind a mai napig nem beszélhetünk még a szemantikus web koráról.

Jelenleg a World Wide Web jellemzően HTML formátumú dokumentumokból épül fel, mely dokumentumok olyan leíró nyelven íródtak, amely elsődleges célja az információ megjelenítése. A HTML jellegéből adódóan a weblapok egy sor leíró szimbólumot (tag- és attribútumnevet stb.) tartalmaznak, melyek arra hivatottak, hogy a böngészőkben az általuk leírt („jelölt”) információt megfelelő módon meg lehessen jeleníteni. A web megjelenését követő első évtizedben a kizárólagos cél nem volt más, mint hogy az információt a felhasználók számára emészthető formában jelenítsék meg. Az emberek olvassák a weboldalakat, értelmezik őket, ugyanakkor a szövegrészek közötti értelmi összefüggések nincsenek úgy ábrázolva, hogy azokat a számítógépek is megértsék.

Épp ezért, a közzétett információkat úgy érdemes ábrázolni, hogy azok ne csak a megjelenítési célokat szolgálják, hanem a számítógépes rendszerek számára is feldolgozhatóak legyenek.

Pontosan ez a szemantikus web célja, az, hogy kialakuljon a gépekhez beszélő Web, azaz a számítógépek még több segítséget adjanak a Weben

található információk kiaknázásához, automatikus (emberi beavatkozástól független) feldolgozásához.

„*A Szemantikus Web nem egy különálló Web, hanem annak kiegészítése, melyben az információ jól definiált jelentéssel bír, ezáltal biztosítva az emberek és a számítógép még hatékonyabb együttműködését*” (Tim Berners-Lee, Hendler, 2001)

Az új technológiák új kihívások elé állítják a szoftverfejlesztőket is. A Web 2.0 korában a webfejlesztés már nem csak arról szól, hogy információkat jelenítsünk meg weboldalakon, mint inkább arról, hogy olyan webalkalmazásokat hozzunk létre, melyek lehetővé teszik a hatékony információmegosztást, biztosítják a gazdag felhasználói élményt, valamint kihasználják a közösség erejében rejlő kollektív intelligenciát.

A fenti felsorolás mindhárom pillére (információ megosztás, felhasználói élmény, intelligencia) fontos, mindegyik területhez kapcsolódóan végeztem kutatásokat, értekezésemben részletesen be is kívánom mutatni az ezzel kapcsolatos eredményeket.

Az Intelligens Web

Bár a Web igen gazdag információforrás, az adatok összegyűjtése, rendszerezése egyre komolyabb feladatot ró a felhasználókat kiszolgáló rendszerek számára. Épp ezért a kutatások és fejlesztések eredményeképpen újabb és újabb technológiák jelennek meg, melyek arra hivatottak, hogy megfelelő tartalommal szolgálják ki az egyes weboldalak látogatóit. Ezen legújabb technológiák összefoglaló neve az intelligens web.

A web intelligencia (WI) kifejezés 2000-ben [5] debütált, *Ning Zhong és társai* szerint az intelligens web a mesterséges intelligencia, a tudásreprezentáció, az adatbányászat, az intelligens ágensek, az intelligens közösségi hálók összefoglaló neve, illetve az ezen területeken szerzett ismeretek, eredmények új közegben történő alkalmazása.

A WI ezen túlmenően új problémák és kihívások megoldására sarkallja az információ technológia és a mesterséges intelligencia szakembereit. A WI technológiák forradalmasítják az információgyűjtés, tárolás, feldolgozás, megjelenítés és megosztás eddig ismert technikáit, módszereit.

Jiming Liu [6] az intelligens web számára négy szintet határoz meg, ahol az információ technológia és a mesterséges intelligencia technikái, technológiái megjelennek.

Vagan értelmezése is hasonló, kutatásai szerint az *intelligens web* kifejezés három technológia, a webbányászat, a szemantikus web és a web megszemélyesítő technológiák összessége [7].

2000 után, szakmai körökben igen népszerű lett az intelligens webbel foglalkozni. 2002-ben megalakult a Web Intelligence Consortium[2], valamint az IEEE Computer Society Technical Committee on Intelligent Informatics bizottsága, és azóta is minden évben az *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology* konferencián jelennek meg a WI technológiával kapcsolatos kutatások.

A népszerűség oka nem véletlen, a WI, azon belül is a web megszemélyesítés egyike azon kutatási területnek, amely remek eszköztárat és gazdasági hasznot biztosíthat az e-business iparágnak. A technológia olyan képességei, mint a felhasználók vásárlási vagy böngészési szokásainak folyamatos figyelése, rögzítése, elemzése, minden eddiginél hatékonyabb eszközt ad a kereskedők kezébe ahhoz, hogy a lehető legpontosabban célozza, személyre szabja az értékesítendő termékét, szolgáltatását.

A felhasználói élmény növelésének egyik legjobb módszere a web megszemélyesítés, hisz segítségével tudjuk elérni azt, hogy minden egyes látogató ugyanazon weblap saját, személyre szabott változatát kapja. A technológiának köszönhetően a webszerverek a weboldalt akár valós időben képesek a vásárló igényeire igazítani, ezáltal biztosítva azt a különleges, egyedülálló felhasználói élményt, mely során a látogató úgy érzi, a weboldal szinte olvas a gondolatában.

A megszemélyesítés témakörével számos kutató foglalkozott már, így A.R Simon és társa [8] a célzott marketing szempontjából, Ning Zhong és társai a wisdom web szemszögéből [9], Su Ho Ha a vásárlói döntésekre gyakorolt hatását vizsgálta [10], míg Wang és Kobsa [11] a személyes adatok védelmét figyelembe véve dolgozott ki megfelelő módszert rá.

2 http://wi-consortium.org

A web megszemélyesítés alapfeltétele a weboldal látogatóról felhasználói profil készítése, majd ezen profilok alapján képes az ajánlói rendszerünk (recommender system) releváns tartalmat szolgáltatni. Az első ajánlói rendszerek már a 90-es évek elején megjelentek, jellemzően az együttműködés alapú szűrés (collaborative filtering) terminológiájának kiterjesztéseként, majd ahogy mind a kutatók, mind a lehetséges felhasználási területek száma nőtt, úgy fejlődött maga a technológia is.

Mára már ide soroljuk az együttműködés alapú [12], a tartalom-alapú (content-based) [13], a tudás-alapú (knowledge-based) [14] és a demográfiai szűrést [15]. Ezen technológiákat a hatékonyság növelése érdekében időről időre kombinálják, így Burke hibrid ajánlói rendszernek [16], Melville és társai Content-Boosted Collaborative Filteringnek [17], míg Sobecki Consensus-based ajánlói rendszernek [18] nevezte el saját megoldását.

Személyes meggyőződésem ugyanakkor, hogy az intelligens web fogalma ma, a mobil eszközök és szenzorok, vagy ha úgy tetszik az Internet of Things (IoT)[3] és az Internet of Everything (IoE)[4] világában már jóval több, mint amit az elmúlt 10-12 évben értettünk a kifejezés alatt. Elismerve az előzőekben felsorolt kutatók munkásságát, engedtessék meg mégis egy saját definíció az intelligens web fogalmára:

Az intelligens web eszközök olyan hálózata, mely eszközök képesek a felhasználókról, a környezetükről és saját állapotukról a lehető legtöbb információt implicit módon összegyűjteni és ezen aggregált információkból olyan következtetéseket levonni, ajánlásokat tenni, melyek a rendszer használói számára személyre és alkalmazási szituációra szabottak.

A fenti definíció pontos megértéséhez és egyben egyfajta általános szabályként történő alkalmazhatóságának bizonyítására következzen két példa.

Probléma 1:

Egy *angol* turista *Debrecen*ben *okostelefon*ja segítségével *éttermet* keres.

Megoldás 1:

Az intelligens webrendszerünk a felhasználó *mobiltelefonjára optimalizált* felbontással és tartalmi részletességgel, *angol nyelven* listázza ki, a turista tartózkodási helyének *500m-es körzetében* fellelhető debreceni *éttermeket*.

3 http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things

4 http://www.cisco.com/web/about/ac79/innov/IoE.html

Eszközök:

Az intelligens webrendszerünk ebben az esetben az alábbi eszközökből áll: 1.okostelefon, 2. GPS műhold, 3. központi szerver számítógép

Működés:

Az okostelefon kapcsolatba lép a GPS műholddal, lekéri az aktuális koordinátákat.

A megkapott információt megküldi a központi szerver számítógépnek.

A szerver számítógép a megadott GPS adatok alapján a központi adatbázisból lekéri az 500m-es körzetben lévő éttermek listáját.

A szerver kapcsolatba lép az okostelefonnal, lekéri annak típusát, nyelvi beállításait, képernyő felbontását.

A megkapott információk alapján az angol nyelvű éttermi anyagot kiválasztva a készülékre optimalizálja a képek és a tartalom méretét, majd megküldi azt a felhasználó mobil eszközére.

Probléma 2:

Egy *autó és utasai balesetet szenvednek*, *mihamarabb segítséget* kell hívnia a gépjárműbe épített automata rendszernek.

Megoldás 2:

A *gépjárműbe épített* intelligens rendszer *felhívja* a *legközelebbi* központi kórházat, közli a baleset *pontos helyszín*ét, az *időjárási körülmény*eket, az *utasok számát, az ütközés sebességét, a gépjárműben bekövetkezett kár mértékét.*

Eszközök:

1. gépjárműbe épített szenzorok, 2. fedélzeti számítógép, 3. GPS műhold, 4. központi egészségügyi adatbázis (szerver), 5. kórházi ügyeleti telefonközpont

Működés:

A gépjárműbe épített szenzorok érzékelik, hogy baleset következett be.

A fedélzeti számítógép lekéri a GPS műholdról a pontos koordinátákat.

A külső hőmérsékleti és csapadék mérő szenzorok folyamatosan rögzített adataiból információcsomagot készít az időjárási körülményekről.

Az ülésekbe épített érzékelők adataiból, valamint a sofőr vezetési stílusából megállapítja az utasok számát, súlyát.

Kapcsolatba lép a központi egészségügyi adatbázissal és a megkapott GPS koordináták alapján lekéri a legközelebbi kórházak telefonszámát.

Meghatározott algoritmus alapján tárcsázza a kórházakat és mind hangcsomagként mind adatcsomagként eljuttatja hozzájuk a rendelkezésre álló információkat.

Problémák

Az intelligens web rendszerekkel kapcsolatos kutatások azonban szinte kizárólag arra irányulnak, hogy a módszer magját képező algoritmusokat minél jobban optimalizálják, ugyanakkor igen kevés szó esik arról, hogy a feldolgozás alapját képező adatokat milyen módszerrel, technológiákkal érdemes összegyűjteni, illetve a végeredményt, a kimeneti adatokat milyen formában érdemes tálalni a felhasználó számára.

Mind az input, mind az output kiemelt jelentőséggel bír, hisz bemenő adatok nélkül a legtökéletesebb algoritmus sem ér semmit, mint ahogy egy rossz, lassú webportál mögött is dolgozhat bármilyen hatékony mesterséges intelligencia, ha a potenciális vásárlók, látogatók inkább menekülnek az oldalról, mintsem hogy böngésszék azt.

Azt gondolhatnánk, hogy mi sem egyszerűbb annál, mint adatokat gyűjteni a felhasználókról, hisz az Internet épp erről, a milliárdnyi adatról szól. Az intelligens rendszerek szempontjából viszont a felhasználókkal és azok viselkedésével kapcsolatos adatok bírnak kiemelt jelentőséggel, melyeket kétféle módon szerezhetünk be: vagy megkérjük a felhasználót, hogy adja meg az általunk kért adatokat (űrlapok, kérdőívek, termékértékelések formájában például) – ez az úgynevezett explicit adatgyűjtés, vagy az intelligens rendszerünk a felhasználó tudta, zavarása nélkül, a háttérben gyűjti az adatokat (böngészési útvonal, kattintások, látogatott weblapok, vásárlási előzmények, IP cím, nyelvi beállítások stb.) - ezt nevezzük implicit adatgyűjtésnek.
A két technológia kombinálható, ám mindkettővel akadnak problémák: az explicit adatgyűjtés nehézkes, a felhasználók egyrészt bizalmatlanok, nem szívesen adnak meg adatokat, másrészt az időt is sajnálják arra, hogy a kedvünkért űrlapokat töltsenek ki.

Az implicit adatgyűjtés az előző problémákat kiküszöböli, továbbá nagy előnye, hogy segítségével hatalmas mennyiségű adatot áll módunkban összegyűjteni, ám a jellegéből adódóan felvet egy igen fontos személyiségi jogi kérdést: gyűjthetünk e a felhasználóról a tudta, beleegyezése nélkül bármilyen

adatot? Egyáltalán, tisztában vagyunk e azzal, hogy abban a pillanatban, amikor megnyitunk egy weboldalt a számítógépünk mögött meghúzódó informatikai rendszer milyen adatokat, információkat gyűjtött és tárolt rólunk?

Webfejlesztés

Egy web-alapú rendszer fejlesztése jóval több feladatot ró a fejlesztőkre, mint egy hagyományos szoftverfejlesztés; a rendszer életciklusa, a fejlesztésének menete, a nyomkövetés és fenntartás, mind mind különbözik a klasszikus szoftverfejlesztéstől. Érthető hát, hogy a tradicionális fejlesztési módszertanok sok esetben a web-alapú rendszerekre nem igazak, pontosabban korrekcióra, kiegészítésre szorulnak. Remekül fogalmazta meg Powell 2000-ben kiadott könyvében a lényeget:

"A webfejlesztés ötvözi az újságkiadást a szoftverfejlesztéssel, a marketinget a számítástechnikával, a belső kommunikációt, a külső kapcsolatokkal, a művészetet a technológiával" [23].

Különböző szakirodalom különböző, alapjaiban mégis hasonló módon definiálja a webfejlesztés (web engineering) fogalmát. Ezek alapján összefoglalva elmondhatjuk, hogy a webfejlesztés nem más, mint módszer a webalkalmazás-fejlesztéssel kapcsolatos tudás fejlesztésére és szervezésére. Leginkább Murugesan és társai által megfogalmazottakkal értek egyet, mely szerint a webfejlesztés a tudományos, mérnöki és menedzsment tudományokkal kapcsolatos ismeretek szisztematikus alkalmazása a web-alapú rendszerek sikeres fejlesztése, üzembe helyezése és fenntartása érdekében [24].

Hozzátéve, hogy a Web engineering még ettől is több diszciplínát ölel fel; ahogy kitűnik Powell jellemzéséből is, a számítás- és informatika tudományokon és mérnöki ismereteken túl a menedzsmenthez, a művészethez, grafikához, valamint az ember-gép közötti kapcsolathoz értő szakemberek éppúgy nélkülözhetetlen elemei a fejlesztésnek, mint a marketing szakemberek.

A web-alapú rendszerek jellemzői

Ahhoz, hogy ezt megértsük, tekintsük át, melyek a web-alapú rendszerek legfontosabb jellemzői, melyek azok az ismérvek, melyek megkülönböztetik őket a hagyományos szoftverrendszerektől. Többen, többször megfogalmazták már ezeket az ismérveket [25,26,27,28], az alábbiakban ezeknek egy leporolt, újra átgondolt, tapasztalataim alapján frissített csoportosítását mutatom be.

Heterogén felhasználók

A webalkalmazásokat rengeteg különböző, - akár milliós nagyságrendű – a rendszer számára ismeretlen felhasználó használja, akiknek eltérőek a rendszerrel szemben támasztott igényeik, az informatikai felkészültségük. Épp ezért a felhasználói felület kinézete, annak kezelhetősége – különös tekintettel az akadálymentesítés irányelveire[5] - és a szolgáltatáspaletta meg kell feleljen ezen színes felhasználói közösségnek, figyelembe véve azt az igen fontos tényt, hogy webrendszerek esetében nincs lehetőségünk képzéseket tartani a rendszer használatáról, szemben a tradicionális szoftverekkel, ahol ez a folyamat elvárás és kívánalom.

A felhasználói élmény kiemelt szerepe

Kiemelt szerepet kap a kinézet és a felhasználói érzet, a webfejlesztés egyik igen fontos szegmense művészeti és marketing munka. Az arculat megtervezése, a felhasználói felület ergonómiájának kialakítása, a tartalom precíz elhelyezése külön kreatív csapatok feladata. A webrendszer egy adott cég, szervezet, termék arca a külvilág felé, jóval nagyobb szerepe van itt a felhasználói élménynek, mint egy klasszikus szoftver esetében. Ha egy webrendszer rosszul működik, vagy nem elégedettek vele a felhasználók, az sokkal súlyosabb következményekkel jár a tulajdonosra nézve, mint egy hagyományos szoftver esetében.

Dinamikus tartalom

A mai webrendszerek dinamikusak, tartalom (adatbázis) alapúak. A fejlesztés fontos része a tartalom létrehozás és a folyamatos tartalomfrissítés biztosításának lehetősége; a legelső élesített állapot után akár már egy óra múlva más a tartalom. Nagyobb rendszerek esetén – lásd eBay vagy Amazon - ez a tartalom akár percenként változik, bővül.

Folyamatos evolúció

A webalkalmazások folyamatosan fejlődnek. Épp ezért lehetetlen komplett rendszer-specifikációt adni a fejlesztési folyamat legelején, hisz a funkciók és szolgáltatások folyamatosan változnak, bővülnek, különösen miután élesítették, használatba vették a webrendszert. A hagyományos szoftverekkel ellentétben,

5 http://www.w3.org/TR/WCAG20/

amelyek előre tervezett, jól meghatározott időközönként esnek át felülvizsgálaton, a webalkalmazásokkal szemben támasztott követelmények és funkcionalitás állandó változása miatt ez a munka folyamatos; legalább annyira szervezési, menedzselési kérdés, mint technikai.

Szoros határidők

A webalkalmazások fejlesztésére mindig rendkívül szoros határidőket szabnak, így a fejlesztőkre gyakorolt nyomás rendkívül nagy. Egy, a klasszikus szoftverfejlesztésnél megszokott, akár egy évnél is hosszabb időre elnyújtott alapos fejlesztési terv itt nem működik; az idő kiemelten fontos tényező.

Gyors technológiai fejlődés

Rendkívül gyors a technológiai fejlődés ezen a területen, az új szabványok, eszközök, nyelvek alkalmazása, beépítése, illetve a korai verziók hibái, kompatibilitási problémai megsokszorozzák a hiba lehetőségét.

Heterogén szoftvertechnológia

Mindezen túl a webfejlesztés során rengeteg különböző technológiát használunk, klasszikus programozási nyelveket, szkript nyelveket, HTML, CSS és XML állományokat valamint ezek kombinációját, adatbázisokat és lekérdező nyelveket, multimédiás elemeket és ezek kezelőszoftvereit, összetett felhasználói felületeket, fejlesztői és segédszoftvereket stb.

Heterogén hardverkörnyezet

Nem elég a technológiai sokszínűség, a kifejlesztett webrendszer rengeteg különböző hardveren fut; különböző képernyő méret, hardvereszköz, hálózati kapcsolat és sebesség.

Kiemelt biztonság és adatvédelem

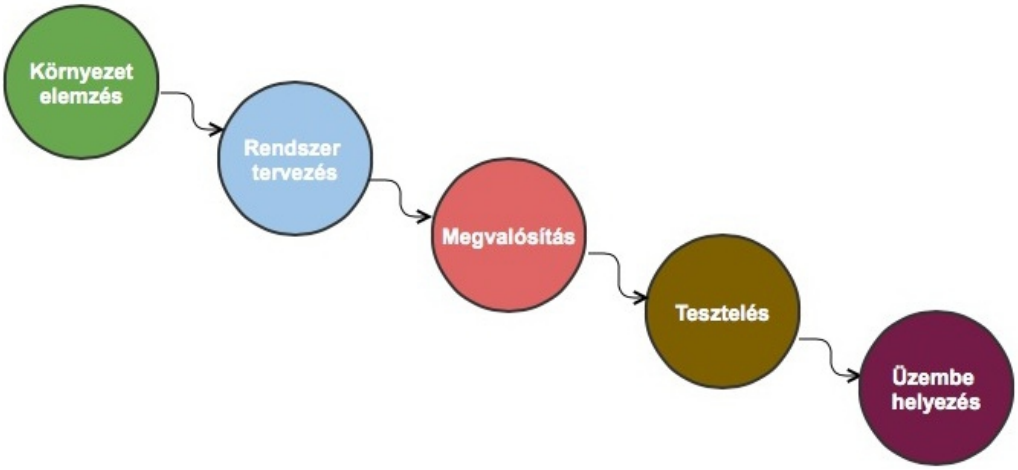
A Web jellegéből adódó hálózatos felépítés, a nagyságrendekkel nagyobb számú azonosítatlan felhasználó jóval több biztonsági és adatvédelmi kérdést vet fel, mint a tradicionális szoftverek.

A webfejlesztés modelljei

Látható tehát, hogy a web-alapú rendszerek sok szempontból különböznek a klasszikus szoftverrendszerektől, igaz ez a fejlesztés és a tervezés folyamatára is.

A szoftverfejlesztési folyamat bonyolult és összetett, az egyes részfeladatokat, illetve az ezek közötti kapcsolatokat modellezni szükséges annak érdekében, hogy egyrészt áttekinthető, másrészt moduláris felépítésű legyen.

A legismertebb és legelterjedtebb a vízesés (waterfall) modell, mely a szakirodalomban először Winston W. Royce 1970-es cikkében jelent meg [29], bár érdekesség, hogy maga a *waterfall* kifejezés az írásban egyetlen egyszer sem szerepel. Sőt épp az ilyen jellegű fejlesztési folyamat hibáit, hátrányait ismerteti.



4. ábra: A vízesés modell

A modell mégis jól alkalmazható abban az esetben, ha a rendszerrel szemben támasztott követelményeket már a fejlesztés legelején tudjuk. Ez természetesen alapos felmérő és kutatómunkát igényel, nem beszélve arról, hogy a megrendelőnek is igen felkészültnek kell lennie az elvárt funkcionalitást illetően, hisz a menet közbeni változtatások rendkívül költségesek olyannyira, hogy bizonyos lépcsőfokok után már gazdaságosabb a legelejéről újrakezdeni a fejlesztést.

Amennyiben viszont alapos és részletes tervvel rendelkezünk, mind a várható

költségek, mind a fejlesztés ideje igen jól becsülhető. Ráadásul a szigorú, tervekhez illeszkedő fejlesztési folyamat kevésbé érzékeny arra, ha valamely fejlesztőnktől meg kell válnunk, a precíz dokumentáció miatt az új munkaerő hamar beilleszthető a csapatba.

A modell jellemzője, hogy a következő fázis addig nem indulhat el, amíg az előző fázis be nem fejeződött. Így egy-egy részfolyamatra delegált fejlesztő, tervező munkája befejeztével más projekten is dolgozhat. Az életciklus utolsó fázisában kerül átadásra a felhasználók számára a rendszer, ekkor derül ki, hogy a szoftver teljesíti e az eredeti specifikációban megfogalmazott követelményeket vagy sem, azt kapta e az ügyfél, amit szeretett volna.

Sajnos a való életben a megrendelő nem tudja teljes pontossággal definiálni a projekt elején, mit is akar; a követelmények menet közben változnak, finomodnak. Épp ezért a vízesés modellt az üzleti webfejlesztésben egyre kevésbé használják, ugyanis ezekre a változásokra a vízesés modell nincs felkészülve; ha valamely fejlesztési fázis lezárult, szinte lehetetlen azon változtatásokat eszközölni.

Egy olyan módszerre van szükség, mely rugalmas, lehetővé teszi azt, hogy menet közben rendszeresen konzultáljunk a megrendelővel, és ha módosításra van szükség, azt a legkisebb költséggel tegye lehetővé.

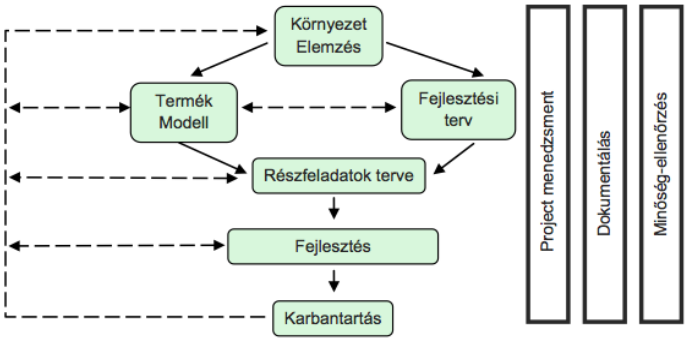
Már Royce cikkében is megjelenik az igény egy új, iteratívabb, interaktívabb módszer alkalmazására, nem véletlen, hogy az évek folyamán megannyi vízesés modell alternatíva, továbbfejlesztett módszertan született.

Bíró külön cikkben [30] ismerteti a fel-fel bukkanó módszertani divathullámok karakterisztikáit, a legújabb hullám még ma is tart és gyökerei egész 1988-ig, Boehm spirál [31] és Gilb evolúciós modelljéig [32] nyúlnak vissza.

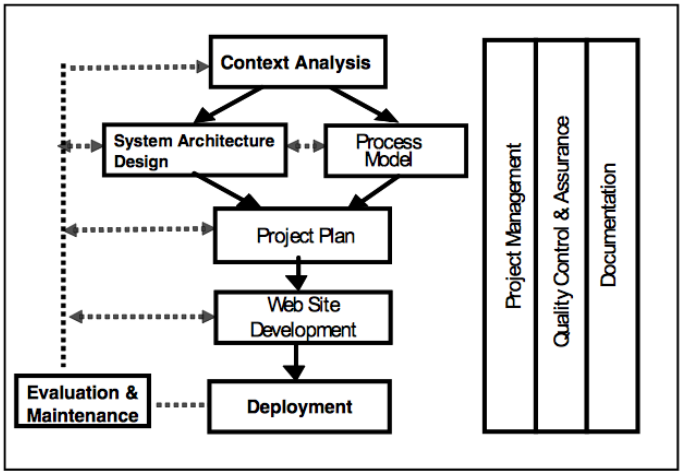
Ez az új trend az agilis szoftverfejlesztés, mely sok szempontból épp az ellenkezője a vízesés modellnek. A fejlesztők, ahogy elkészült egy kisebb modul, átadják tesztelésre a felhasználónak, aki így kipróbálhatja, pontosíthatja az igényeit, az új igényekre gyorsan és hatékonyan reagálva módosítják azt, majd ismét egyeztetnek a megrendelővel; a folyamatos konzultáció és iteráció az agilis szoftverfejlesztés alapja. A módszer garancia arra, hogy a felhasználó elégedett, azt kapja amit szeretett volna, még ha a fejlesztés elején nem is tudta kellő alapossággal megfogalmazni igényeit.

Az agilis módszerhez hasonló fejlesztési folyamatot ismertet Adamkó (5. ábra) [33], mely a Murugesan - féle (6. ábra) [34] folyamat korszerűbb

változata és ma is megállja a helyét. Ugyanakkor ahhoz, hogy a mai, intelligens és reszponzív webrendszerek által támasztott igényeknek megfeleljünk, egy új, részletesebben kifejtett hibrid módszerre van igény.



5. ábra: Adamkó - féle webfejlesztés folyamat



6. ábra: Murugesan - féle webfejlesztés folyamat

A saját módszer indokoltsága

Pusztán az agilis módszertani divathullámot meglovagolva mondhatnánk azt, hogy a mai kor fejlesztési folyamatának egyértelműen ezt választjuk, de a módszernek vannak hátrányai. A folyamatos konzultáció remek eszköz, de időigényes, sőt, ha a megrendelő épp nem ér rá, hátráltatja a fejlesztést. Az állandó revízió és változtatás lehetősége miatt jóval több szakembernek (grafikus, rendszertervező, front-end, back-end programozó) kell rendelkezésre állnia egyidőben, szemben például a vízesés modellel. Mindezek mellett a várható költségek és befejezési határidő is nehezen becsülhető.

Ésszerű feltételezés, hogy a két rendszer előnyeit kombinálva hatékonyabb metódust kínálhatunk a webfejlesztésre. A hibrid rendszer kialakításában nagy segítséget jelentett az üzleti élet szereplőivel való konzultáció, a saját fejlesztői tapasztalat, mely alapján azonosítani tudtuk azokat a hibákat, kritikus pontokat, amelyek egy nem megfelelő módszertan alkalmazása, vagy épp bármilyen módszertan figyelmen kívül hagyása miatt jelentkezhetnek egy alkalmazásfejlesztési folyamat során.

Mindezeken túl ismét hangsúlyozni szükséges a felhasználói interfész, az arculat és a design fontosságát, így a kifejlesztett hibrid módszer már erre vonatkozóan is tartalmaz fejlesztési modellt. Az eddigi modellek elegánsan átléptek ezen a ponton, vagy épp egybemosták a rendszertervvel, holott a megrendelő és a majdani felhasználók számára - érthető módon – a kinézet és érzet, az ún. „look & feel” kiemelt jelentőséggel bír.

Adaptív fejlesztési módszer

Felhasználói felület (UI) tervezése

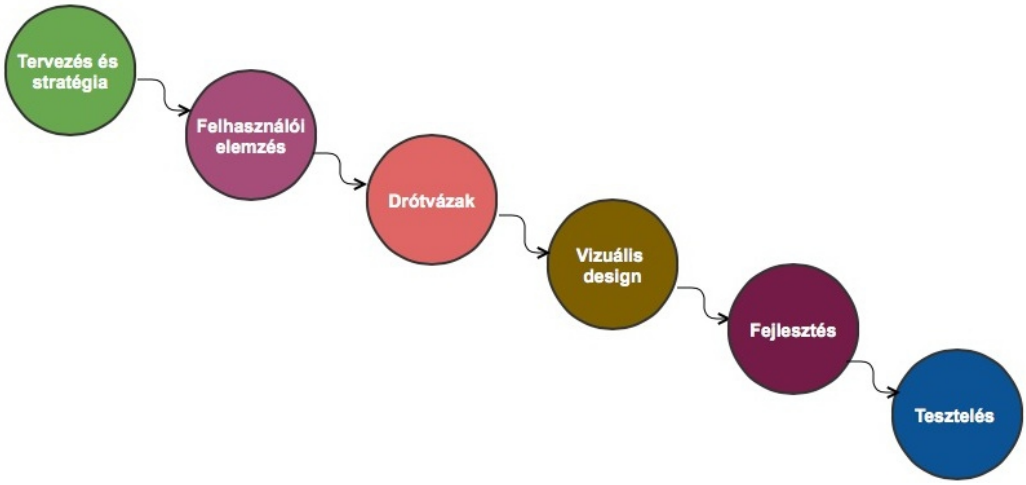
Hagyományos UI fejlesztése

Az eddigi jól bevált üzleti gyakorlat szerint egy webdesign készítés az alábbi folyamat alapján készült.

Az igényfelmérés és a versenytársak weboldalainak elemzése után a megrendelő elmondta elképzeléseit a kinézetről, a színvilágról, a funkciókról.

Ezek alapján készült egy oldaltérkép, egy drótváz (wireframe), majd ezek alapján 2-3 PhotoShop grafikai terv.

Ezeket a megrendelő átnézte, kiválasztotta a neki megfelelőt, vagy épp módosítást kért, végül lett egy elfogadott designterv, amit aláírt. Ebből aztán készült egy HTML template, mely ezután átkerült a szoftverfejlesztőkhöz (7. ábra).



7. ábra: Felhasználói interfész klasszikus tervezés folyamata

Ismerős a folyamat, igen, klasszikus vízesés modell. Adaptív, illetve reszponzív webrendszereknél ez a folyamat nem életképes. Nem lehet minden egyes készüléktípusra elkészíteni a designtervet és azt egyesével elfogadtatni a megrendelővel. Ezt a procedúrát nem szabad és nem is kell végigerőltetnünk; sem magunkon, sem a megrendelőn. Helyette egy gyorsabb, iteratív módszert alkalmazunk.

Valódi, az üzleti életből vett tervezési folyamatot ismertet Viljami [35], valamint Boulton [36], melyek kiindulási alapot szolgáltattak az alábbi, általam kifejlesztett folyamat megalkotására.

Reszponzív UI fejlesztés

A reszponzív felületek korában a követelményelemzés és információgyűjtés után az alábbi lépéseket valósítjuk meg.

Vázlat

Az információgyűjtés követően készítünk egy vázlatot (sketch), leginkább papír és ceruza segítségével, ám jó szolgálatot tesz a Zurb Responsive Scetchsheet[6] szoftvere is. A vázlatok gyors, szabadkezi rajzok, melyek nem a végtermék modellezésére hivatottak, hanem inkább egyfajta alapkőként funkcionálnak a további tervezési folyamatban. Mivel rendkívül gyorsan elkészül, remek eszköz arra, hogy a hirtelen támadt ötleteket vászonra vessük, azt megmutassuk a felhasználónak. A vázlat és a drótváz közötti különbséget remekül ecseteli a UXMovement cikke [37],

Drótváz (opcionális)

A vázlat elkészülte után jöhet a drótváz, mely a leendő interaktív felhasználói felület struktúraterve. Ezt már nevezhetjük vizuális modellnek, bár sokan - főleg kis- és közepes méretű projektek esetén - kihagyják ezt a lépést. Az érvelésük jogos, a vázlat gyors, a prototípus lassú, ám interaktív, informatív. A drótváz a kettő közötti, se nem gyors, se nem elég informatív a megrendelő számára.

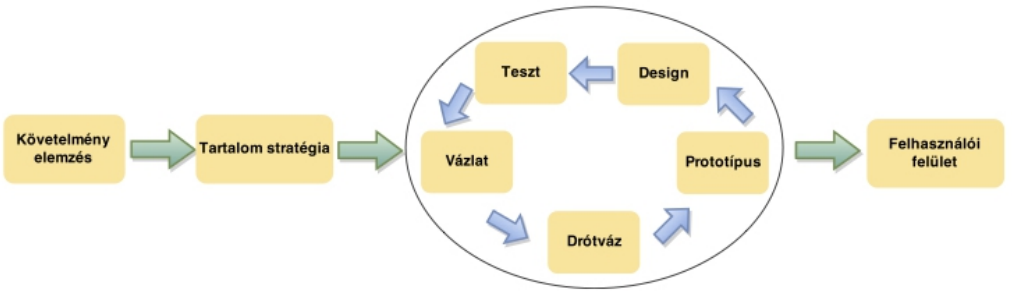
Prototípus

Ezen a ponton HTML, CSS kombinációval a vázlat alapján elkészül egy, böngészőben és legfőképpen a különböző típusú és felbontású mobilkészülékeken megtekinthető kinézet. Természetesen ezen a ponton már alap grafikai elemek is megjelennek, de a részletes kidolgozás a következő lépésben valósul meg.

Design

Jellemzően ez az a pont, amikor Photoshop, FireWorks vagy egyéb grafikai szoftver segítségével elkészülnek a grafikai tervek. A részletek kidolgozása projektfüggő, hisz az előző pontokhoz hasonlóan ez is egy iteratív lépés, ahogy majd ez a 8. ábrán látható.

6 http://zurb.com/playground/responsive-sketchsheets



8. ábra: Reszponzív UI fejlesztés

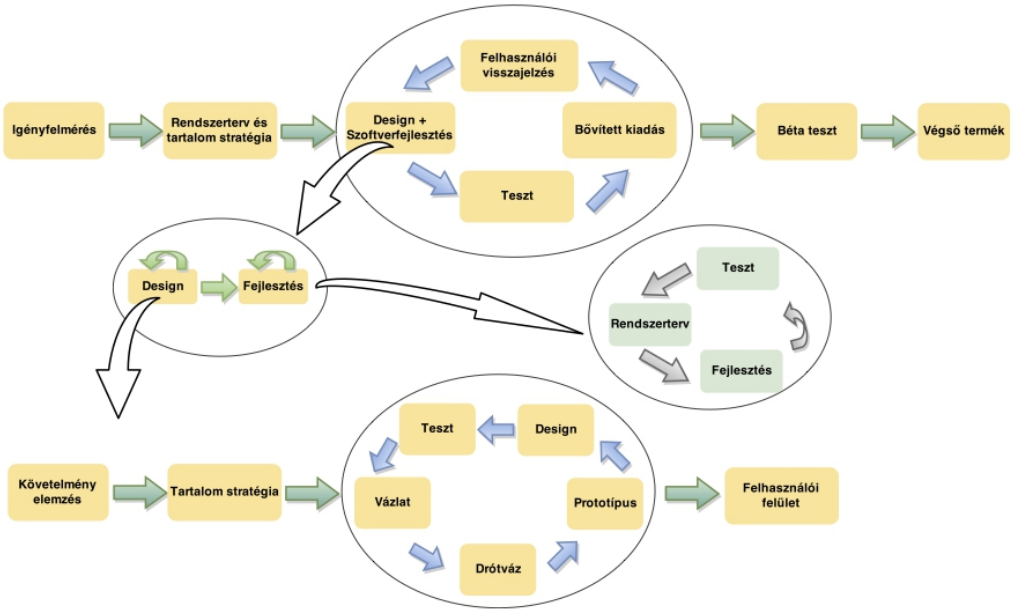
Jól látható, hogy a fejlesztési folyamat számos pontján visszaköszönnek az agilis szoftverfejlesztés elemei. Így mind a felhasználói felület, mind a teljes webrendszerünk fejlesztési folyamatmodellje egyfajta kombinált megoldásért kiált.

Az új rendszerfejlesztési módszer

Akármennyire is nehézkesnek tartjuk a vízesés modellt, segítségével igen jól becsülhető a projekt várható költsége és időtartama. A megrendelővel kötött szerződés szempontjából ez a két tényező kiemelt fontosságú, már a projekt legelején rögzítenünk kell őket. Az sem árt, ha minél több információt sikerül begyűjtenünk a rendszer-specifikáció, a rendszerterv elkészítéséhez; egy alapos terv nagymértékben megkönnyíti a jövőbeni munkát. Ugyanakkor az agilis módszer flexibilitása elengedhetetlen, a való életben a kezdeti funkcionalitás bővül, változik.

Új modulok, új menüstruktúra, új kinézet: a konkurencia új szolgáltatást kínál, ezért építsük be mi is; a marketingesek menet közben más tartalomelrendezést találtak ki, alakítsuk át; az uralkodó szín mégse a narancssárga, hanem a világoszöld legyen, készüljön új látványterv; mégiscsak legyen geo-érzékeny hírlevél feliratkozás és hozzá tartozó adminisztrációs felület. Mind-mind gyakorlatból vett példa, melyre a fejlesztési folyamatunknak fel kell készülnie.

A fentiek alapján az alábbi folyamatmodellt alkottam meg a mai modern, adaptív webrendszerek fejlesztéséhez.



9. ábra: Az adaptív fejlesztési modell

Lássuk, hogy a fejlesztés egyes fázisai mit is takarnak:

Igényfelmérés

A követelményelemzés, az igényfelmérés a webfejlesztés első lépése, ezen a ponton az alábbi információkat gyűjtjük be:

1. Azonosítjuk a projektben érintetteket és felmérjük igényeiket, tapasztalatukat. Megismerjük a webrendszer leendő felhasználóit, azok lehetséges létszámát, ehhez méretezzük majd a rendszert.
2. Meghatározzuk a webrendszer által nyújtandó szolgáltatásokat. Ez a lépés képezi majd a menü és navigációs rendszer kialakításának alapját, a webportál különböző moduljai is ezen információk alapján kerülnek meghatározásra.
3. Meghatározzuk milyen információ jelenjen meg a weboldalon, hogyan érhetőek el ezek az információk és milyen gyakran fog ez az információ változni.

4. A megrendelő igényeit felmérjük a kinézet, a megjelenés, a biztonság és a működtetés terén.
5. Megismerkedünk hasonló weboldalakkal, a versenytársak webrendszereivel, tanulmányozzuk azok szolgáltatásait, erősségeit, korlátait. Ez a lépés különösen hasznos a 3. pontban meghatározott információk begyűjtésében, a megrendelő jóval hamarabb tudja megfogalmazni igényeit, ha lát konkrét mintákat, ötleteket.

Rendszerterv és tartalomstratégia

A rendszerterv készítésekor meghatározzuk a rendszer komponenseit és a köztük lévő kapcsolatokat. Itt írjuk le a hálózati és szerverkapcsolatokat (legyen az kliens-szerver, web-, alkalmazás- vagy adatbázisszerver), illetve a rendszer különböző moduljait és azokat a funkciókat és szolgáltatásokat, melyeket az egyes modulok megvalósítanak. Ezen a ponton lép be valamely tervezési minta alkalmazása, itt határozzuk meg a rendszer architektúráját, itt válik el egymástól itt kerül kialakításra a felhasználói felület, az adatmodell és az alkalmazás logika. Az agilis módszertani jellemzők miatt a rendszer kezdeti funkciói, szolgáltatási a későbbiekben változhatnak, ám a fejlesztés szempontjából releváns tervezési minta, az alkalmazandó hardver és szoftvertechnológiák, keretrendszerek már itt rögzítésre kerülnek.

A tartalomstratégia a reszponzivitás miatt külön figyelmet érdemel; nem azért, mert a mobileszközök számára más tartalmat kell szolgáltatni, mint az asztali számítógépek számára, sőt! A reszponzív fejlesztés egyik aranyszabálya: A mobil felhasználók ugyanazt a tartalmat akarják látni, mint az asztali felhasználók. Hibás az az elképzelés, hogy egy webrendszer mobil nézete, a normál webportál butított verziója. Mint ahogy az is, ha valaki a 960 pixel széles felhasználói felületre tervezett weboldalát gondolja egy az egyben megjeleníttetni a 320 vagy 480 pixel felbontású okostelefonon.

A helyes és indokolt tartalomstratégia a “mobile-first” irányzat, azaz – bár elsőre szokatlan és az eddigi gyakorlattól eltér – először mobil platformra készítsük el a terveinket. A módszernek két fontos előnye van, az egyik üzleti, a másik tervezési:

Egyre növekvő mobilhasználat

A Business Insider 2013 novemberi felmérése [38] szerint az online eszközök 60%-a okostelefon vagy táblagép, az internetforgalom 20%-át teszi ki a mobil adatforgalom, ezen felhasználók 20%-a már mobileszközön intézi vásárlásait; ráadásul mindhárom szám egyértelmű, növekvő tendenciát mutat.

Jobban átgondolt tartalom

A kisebb képernyőméret segít abban, hogy a megjelenítendő információk, szolgáltatások fontossági sorrendjét alaposabban átgondoljuk, a méretbeli korlátok rákényszerítenek minket arra, hogy a kevésbé lényeges elemeket háttérbe szorítsuk. Egy letisztult, lényegre törő felhasználói felület nagyobb marketing-értékkel bír, áttekinthető, gyors és a legfontosabb: a mobil felhasználó elégedett. A letisztult tartalom ugyanakkor nem feltétlenül jelent kevesebb, mint inkább sokkal jobban szegmentált információt, sőt ezen tervezési módszer nemcsak a mobil jellegű platformokra alkalmazható, hanem összhangban van az akadálymentes és egyetemes tervezés egyre általánosabb elveivel is. Természetesen egy nagyobb képernyőn több grafikai elem, nagyobb kép, részletgazdagabb diagram, bővebb szöveges leírás helyezhető el, ám ha ezt az egyszer már jól átgondolt tartalmat egészítjük ki, színesítjük, az érdemi információ mindvégig a középpontban marad.

Iteratív Design és Szoftverfejlesztés

A felhasználói interfész fejlesztésének lépéseit az előző fejezetben ismertettük, az ott alkalmazott iteratív folyamat része a komplett fejlesztési folyamatunknak. Miután elkészült valamelyik felhasználói felületre a grafikai terv a HTML template a rendszertervekkel együtt átkerül a front-end és back-end fejlesztőkhöz. Itt rögtön meg is érkeztünk egy szintén aktuális problémához. Hogyan lehet mind elméleti megközelítéssel, mind gyakorlati technológiával összekötni a meglévő kliens oldali és szerver oldali keretrendszereket. Erre kínál megoldást disszertációnk 6. fejezete.

Akárhogy is, a szoftverfejlesztés ezen a szinten szintén az agilis módszertant követi, a rendszerterv alapján elkészül egy szoftvermodul, melyet teszt, majd újabb finomítás követ; a folyamat ciklikus (9. ábra).

Teszt, Bővített kiadás, felhasználói visszajelzés

Miután összeállt a design és a programkód, jöhet a tesztelés, majd a felhasználó számára mutatható újabb, bővített kiadás. A megrendelő rögtön tesztelheti is, észrevételezheti az eddig elkészült rendszert, funkcionalitást, majd a visszajelzések alapján újabb korrekció, újabb iteráció következik. Érdemes megjegyezni, hogy itt 1-2 hetes ciklikusságról beszélünk, azaz rendkívül gyors a fejlesztési folyamat. A megrendelő elégedett, egyrészt mert folyamatosan kikérjük a véleményét, másrészt mert látja, hogy ütemesen fejlődik a megálmodott rendszere.

Béta teszt, végső termék

Miután minden rendszerelem a helyére került, jöhet a béta tesztelés, a minőség-ellenőrzés, apróbb hibajavítások, majd piacra kerülhet a végső termék. Ez a lépés nyilvánvalóan nem intézhető el két mondattal, ugyanakkor – ahogy az ábrán is látszik – itt már ismét visszatértünk a szekvenciális vízesés modellhez, így ennek a pontnak az ismertetésétől terjedelmi korlátok miatt eltekintünk.

A teljes munkafolyamat koordinálása és irányítása kiemelt projektmenedzseri és csapatmunkát igényel, így a klasszikus szoftverfejlesztési módszertanokhoz képest ez a módszer lényegesen több időt, energiát, kvalitást követel meg.

Integrált tervezési minta

A mai webalkalmazások fejlesztése megfelelő tervezési minta nélkül nem lehetséges, hisz ma már egyszerre kell ellátni a gazdag kliens oldali programozási feladatokat a megszokott szerver oldali alkalmazásfejlesztéssel. Akár a kliens oldali akár a szerver oldali programozási munkáról beszélünk, a munka volumene megköveteli a tervezési minták használatát. Halmozottan igaz ez egy komplex webalkalmazásra, ahol a kliens-szerver oldali fejlesztés szükségszerűen elválaszthatatlan kapcsolatban áll egymással.

A webes alkalmazásfejlesztés ugyanakkor sok szempontból különbözik a hagyományos szoftverfejlesztési módszerektől, így mindenképp egy új módszert, egy új architektúrát kell kínálnunk a mai kor webfejlesztőinek, webes rendszertervezőinek.

MVC

Nagyobb lélegzetű projektek esetén a tervezési minták használata elengedhetetlen. Számos tervezési minta létezik, ezek közül a legnépszerűbb mind asztali, mind webes környezetben az MVC.

A Modell-Nézet-Kontroller (Model-View-Controller - MVC) tervezési minta nem mai találmány, Trygve Reenskaug már 1979-ben szükségesnek látta a minta életre keltését, akkor még Thing-Model-View-Editor felépítésben, mely kissé átdolgozott változatát a Smalltalk-80-ba már Model-View-Controller néven implementáltak [39]. A Smalltalk-kal jelent meg és azóta számos variációja látott már napvilágot.

Az MVC tervezési minta egy alkalmazás objektumait az alkalmazásban betöltött szerepük alapján három különböző csoportba sorolja: modell, nézet vagy kontroller. A minta azonban nem csak az objektumok alkalmazásban betöltött szerepét definiálja, hanem azt is, hogy azok miképp kommunikálhatnak egymással.

Egy adott MVC objektum csoportot sokszor rétegnek nevezünk, például a modellek csoportját modell rétegnek (modell layer).

A webalkalmazások fejlesztése során népszerű tervezési minta az MVC, használata számos előnyt hordoz. Az MVC-vel felruházott alkalmazások objektumai sokkal inkább újrafelhasználhatóak, az interfészeik pontosabban definiáltak, maga az alkalmazás jóval könnyebben bővíthető, mint más alkalmazások.

Modell

A Modell objektum alatt egyszerre értjük az adatokat, amin az alkalmazásunkban dolgozunk, valamint azon logika és számítási folyamatok definícióját, melyek feldolgozzák, módosítják, manipulálják ezen adatokat. Például egy modell objektum reprezentálhat egy számítógépes játékbeli karaktert vagy egy kapcsolatot a telefonkönyvben. A modell objektum természetesen egy-egy vagy egy-sok kapcsolattal kapcsolódhat más modell objektumokhoz is. Egy jól megtervezett MVC alkalmazásban a legtöbb adat, mely az alkalmazásunk perzisztens állapotának része (lehet ez az állapot tárolva akár fájlokban akár adatbázisban) a modell objektum része kell hogy legyen azután is, hogy az az alkalmazásunkban betöltődött.

Mivel a modell objektumok reprezentálják egy adott problématerülethez

kapcsolódó tudást és tapasztalatot, ezek aztán később egy hasonló probléma, feladat alkalmával újra felhasználhatóak.

Szabványos esetben a Modell objektum nem áll direkt kapcsolatban a Nézet objektumokkal. Ez fontos kívánalom, hisz alapesetben az adat és a nézet igen szoros kapcsolatban áll egymással. A nézet objektumok jelenítik meg például az adatokat, a nézet objektum teszi lehetővé, hogy a felhasználói felületen a felhasználó módosítsa ezeket az adatokat.

A helyes folyamat azonban az, hogy a nézet rétegbeli felhasználói műveletek, - melyek létrehoznak vagy módosítanak adatokat - a kontroller objektumon keresztül kommunikálva eredményezik a modell objektum létrehozását vagy frissítését. Másik irányban is ugyanez a helyzet; amikor egy modell objektum módosul (például adat érkezik a hálózati kapcsolaton keresztül), értesíti a kontroller objektumot, mely azután frissíti a megfelelő nézet objektumokat.

Nézet

A nézet objektum tehát egy olyan objektum az alkalmazásunkban, melyet a felhasználók láthatnak. A nézet objektum tudja, hogyan rajzolja meg magát és hogyan reagáljon a felhasználói műveletekre. A nézet objektumok legfőbb feladata, hogy megjelenítse az alkalmazás modell objektumaiból származó adatokat és lehetővé tegye ezen adatok szerkesztését. Ennek ellenére - ismételten hangsúlyozzuk- , hogy egy MVC alkalmazásban a nézet objektumok nincsenek közvetlen kapcsolatban a modell objektumokkal.

Mivel ugyanazon nézet objektumokat rendszeresen, újra ás újra felhasználjuk különböző alkalmazásainkban, rengeteg előre megírt keretrendszer, könyvtár, osztály áll a fejlesztők rendelkezésére.

A Nézet objektumok a Kontroller objektumokon keresztül szereznek arról tudomást, hogy egy Modellbeli adat változott és frissíteni kell a felhasználói felületen a tartalmat, illetve szintén a Kontroller rétegen keresztül tájékoztatja a Modellt, ha például egy szövegbeviteli mezőben a felhasználó begépelt egy nevet vagy egy email címet.

Kontroller

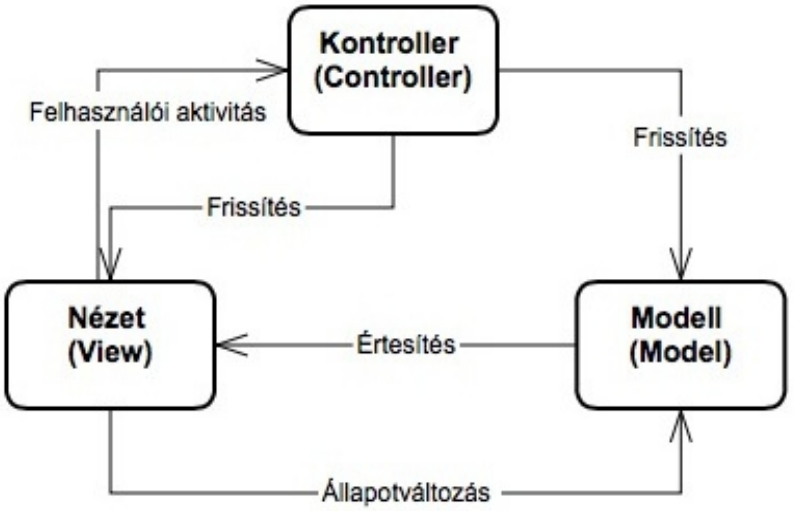
Amint az előzőekben láthattuk a Kontroller objektum átviteli (és transzformációs, koordinálási, szinkronizáló, stb.) szerepet játszik a modell és a nézet objektumok között. A Kontroller réteg feladata, hogy a nézet és a modell rétegbeli változásokat egyik és másik irányba is közvetítse. A Kontroller elláthat továbbá beállítási és koordinálási feladatokat is az alkalmazáson belül,

valamint menedzselheti más objektumok életciklusát is.

A Kontroller objektum értelmezi a Nézet objektumon végrehajtott felhasználói műveleteket, és az új vagy módosított adatokat továbbítja a Modell réteg felé. Amikor a Modell objektumok változnak, a Kontroller objektum továbbítja az új Modell adatot a Nézet objektumnak, mely aztán megjeleníti azokat.

Az MVC architektúra nagy előnye, hogy egyazon Modellhez és Kontrollerhez több Nézet is tartozhat, így webes környezetben kifejezetten jól használható, amikor különböző felbontású készülékeken (számítógép, tablet, mobiltelefon) akarjuk ugyanazt a tartalmat megjeleníteni. Természetesen igaz ez a legújabb trend, a mobil alkalmazásfejlesztés területén is, hogy csak egy példát említsünk, iOS rendszerek fejlesztésénél az Xcode fejlesztői környezet alapértelmezésben legyártja az iPad és iPhone készülékekre méretezett Nézeteket StoryBoard-ok formájában, míg az őket kiszolgáló Modell és a Kontroller ugyanaz marad.

Ugyanakkor nem hagyhatjuk figyelmen kívül a tényt, hogy az eredeti Smalltalk-ban megfogalmazott MVC architektúra engedett bizonyos kommunikációt a Modell és a Nézet között (10. ábra).



10. ábra: A hagyományos MVC architektúra

A felhasználó valamilyen műveletet végez a Nézeten, a Nézet erről egy esemény formájában értesíti a Kontrollert. A Kontroller az esemény hatására – az alkalmazásunk logikájától függően – vagy a Nézetet frissíti, vagy a Modellt szólítja meg, hogy változtassa meg az állapotát. A Modell megváltoztatja állapotát, majd a tényről értesíti a Nézetet, a Nézet pedig lekéri a megváltozott Modell-állapotot.

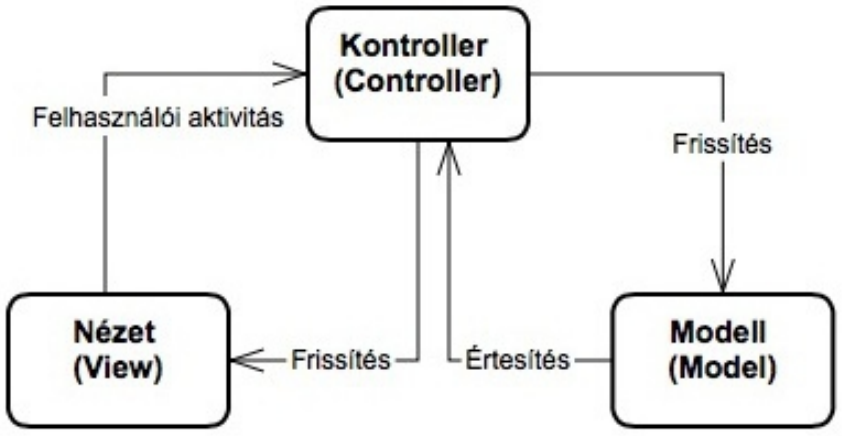
Igen sok mai publikáció is ezt, a hagyományos értelemben vett MVC architektúrát ismerteti, használja, ami igazság szerint az alapkövetelményeket teljesíti is. Hisz a Nézet nem hajt végre módosítást a Modellen, a Modell sem fér közvetlenül hozzá a Nézet tartalmához, ez mindvégig a Kontroller feladata.

Az eredeti MVC minta hiányosságai

Ugyanakkor ezen tervezési minta kapcsán mégiscsak felvetődik egy probléma, ami leginkább az újrafelhasználhatóságot érinti. Egy MVC alkalmazás során a Modell és a Nézet objektumok az alkalmazás leginkább újrahasznosítható elemei. Szinte kizárólag a Nézet objektumokkal találkozik a felhasználó, számára a Nézet jelenti magát az alkalmazást. A következetes, kiszámítható és megszokott működés elengedhetetlen, ez pedig megköveteli az elemek nagymértékű újrafelhasználhatóságát.

A Modell objektumok egyértelműen egy adott problématerület adatait és azzal kapcsolatos folyamatokat fognak egységbe, itt még inkább indokolt, hogy újra fel tudjuk használni az egyszer már megírt, elkészített Modelljeinket más alkalmazásokban is. Mindezek figyelembevételével a tervezési munka során a legjobb, ha a Modell és Nézet objektumokat egymástól függetlenül kezeljük, ez nagymértékben növeli az alkalmazásunk rugalmasságát, és a komponensek újrahasznosítását.

Ennek megfelelően az MVC tervezési minta egy újfajta, az Objective-C Cocoa keretrendszerében használt [40] változatát érdemes alkalmaznunk a mai modern web alapú fejlesztéseink során is, hisz a reszponzív, intelligens web korában egyazon Modellre és Kontrollerre épülve akár több száz Nézet is létezhet, vagy épp több különböző Modellhez tartozhat egyazon Nézet.



11. ábra: A Cocoa keretrendszer féle MVC tervezési minta

Az elsőre példaként említve egy intelligens webportált, amely képes minden egyes felhasználója számára különböző felhasználói felületet nyújtani a készüléktípusától, egyéni preferenciájától, földrajzi tartózkodásától függően. Míg a második esetben egy okostelefonra megírt webes repülőjegy foglalási rendszer – ilyen a Kayak[7] vagy a SkyScanner[8] - különböző, egyre bővülő Modellekből - a világ különböző online foglalási rendszereiből - gyűjti ki egyetlen Nézetre, felhasználói felületre a legolcsóbb, legjobb ajánlatokat.

Meg kell jegyeznünk, hogy az eredeti Smalltalk MVC koncepció szerint a Kontroller elsődleges feladata az, hogy reagáljon a felhasználó eseményekre, azaz mintegy köztes közegként vesz részt a a felhasználó és az alkalmazás között, a Modell frissítése, mint funkcionalitás melléktermék. Ma azonban a Kontrollert a legtöbb implementációban úgy használjuk, mint a Nézetet a Modelltől elválasztó szeparátort, ami eredetileg a Megfigyelő (Observer) feladata volt.

Természetesen az MVC bevezetése óta eltelt közel 35 év, a tervezési minta számos új, átértelmezett változata jelent meg, igazodva a mai modern szoftverfejlesztési igényekhez. Ilyen módosított, némileg átkeresztelt tervezési irányzat a Modell-Nézet-Megjelenítő (Model-View-Presenter MVP).

7*http://www.kayak.com*
8*http://www.skyscanner.com*

MVP

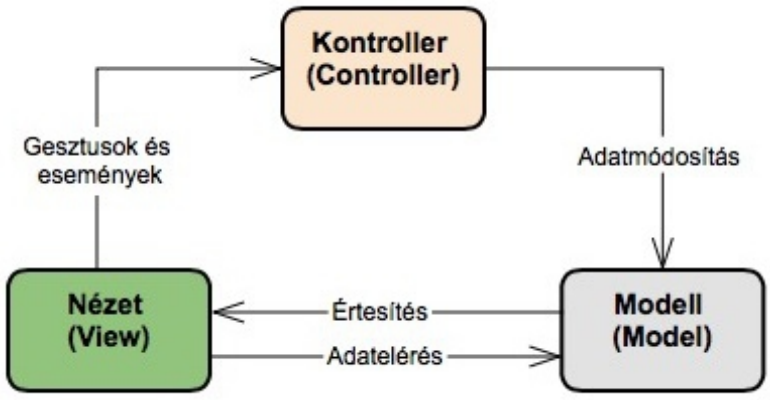
Az MVP az MVC átdolgozott változata, a különbség megfogalmazása azonban nem egyszerű, hisz MVP néven több, egymástól különböző tervezési minta létezik.

Az eredeti MVP mintát Mike Potel - a később IBM által felvásárolt - Taligent Inc. technológia vezetője fogalmazta meg 1996-ban Taligent Programozási Modell néven, mely aztán a szerkezeti felépítéséből MVP néven vált ismertté [41].

Taligent MVP

Potel az eredeti Smalltalk MVC mintát vette alapul, ahol a GUI elemek közül egy szövegbeviteli mezőt hozva példaként, a mezőbe előre beírt szöveg a Modell, a Nézet komponens megkapja az adatot a Modelltől és meghatározza, hogy az az adat miképp jelenjen meg a képernyőn, például egy szövegbeviteli mező és az abba beleírt szöveg képében. A Kontroller feladata annak meghatározása, hogy a felhasználói interakciók, események miképp befolyásolják a Modellbeli adatok változását. Ilyen felhasználói művelet a szövegbeviteli mezőben lévő szöveg módosítása, új adat felvitele. A kör akkor zárul be, amikor a Modell értesíti állapotváltozásáról a Nézetet, mely ezzel tudomásul vette, hogy újra kell rajzolnia a felhasználói felületet.

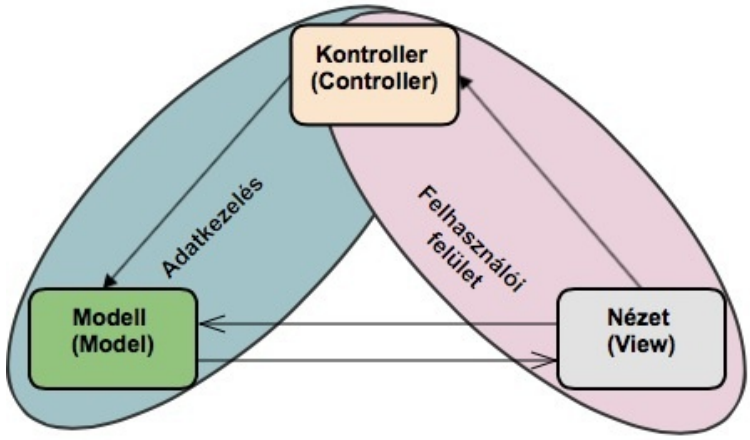
A Smalltalk programozók a GUI objektumaikat és osztályaikat az előbb említett MVC absztrakciónak megfelelően alakították ki és használták (12.ábra).



12. ábra: A Smalltalk MVC

Potel célja az volt, hogy ezt az MVC felépítést még inkább cizellálja, az egyes komponenseket finomítsa, részletesebben tárgyalva, így lehetővé téve, hogy nagyobb, komplexebb projektek fejlesztése során is sikeresen használható legyen.

Első lépésként formálisan is kettéválasztotta a fejlesztési folyamat során egyébként is elkülönülten kezelt Modellt a Nézet-Kontroller párostól (13. ábra). Ez utóbbinak a Megjelenítés (Presentation) nevet adta, ezáltal szétválasztva az adatkezelést a felhasználói felülettől. Ez a két fő terület ugyanis, amivel a programozóknak meg kell birkózniuk, hogyan kezeljék az adatokat illetve hogyan jelenítsék meg azokat.



13. ábra: Kettéválasztott Modell és Nézet-Kontroller

Mivel az adatkezelés, menedzselés igen összetett folyamat, ezért a Modell komponens általánosítása, bővítése elengedhetetlen Potel modelljében (14. ábra).

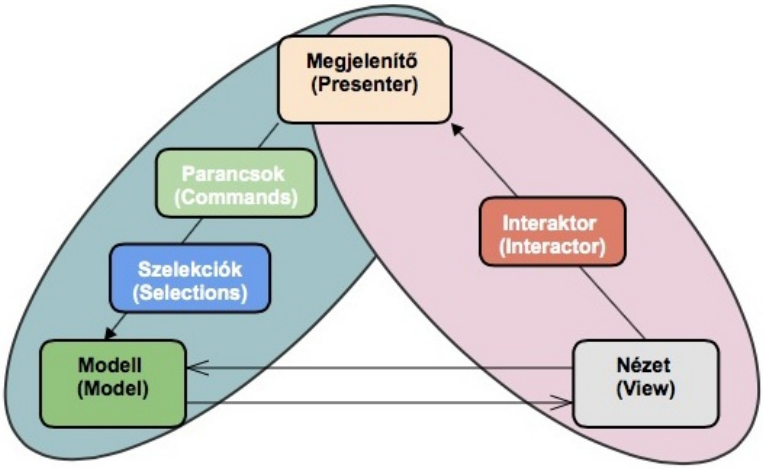
Három fő kérdést fogalmazott meg, mely megválaszolására különböző objektumokat, adatkezelési rétegeket hozott létre a Modellen belül:

- Mi az adat? - Erre a Modell ad választ.
- Hogyan határozom meg az adatot? - Erre a Selections réteg ad választ.
- Hogyan változtatom meg az adataimat? - Erre pedig a Commands réteg a megfelelő.

A Modell tartalmazza ez esetben magát az adatot és az üzleti logikát.

A Szelekció (Selections) komponens határozza meg, hogy a Modellből az adat mely szeletén akarunk műveletet végezni. Ennek eredménye egy adatsor, oszlop, vagy egyetlen elem, mely megfelel az adott feltételnek.

A Parancsok (Commands) határozzák meg, hogy az adaton milyen műveletet lehet végrehajtani. Törlés, beszúrás, módosítás, mentés, nyomtatás és így tovább.



14. ábra: A Taligent MVP

A programozási munka másik nagy fázisa a felhasználói felület kialakítása, működtetése, összekötése a Modellel. Így felhasználói oldalon is megfogalmazódik pár kérdés, melyre a programozóknak választ kell adniuk.

- Hogyan jelenítsem meg az adatokat? - Nézet
- Hogyan hatnak a felhasználói események az adataimra? - Interactor (Interaktor)
- Hogyan kapcsoljam a teljes rendszert össze? - Megjelenítő (Presenter)

A Nézet a Modell vizuális leképezése, felhasználói felületet, azaz a képernyőt, a rajta megjelenő GUI elemeket értjük ezalatt.

Az Interaktorok (Interactors) azok a komponensek, melyek meghatározzák, hogy a felhasználói események miképp feleltethetőek meg a Modellbeli adatok változtatására képes műveleteknek. Interaktor például egy egérmozgás, egy billentyűzet leütés, a drag&drop vagy épp menüelemek vagy checkboxok kiválasztása.

A Megjelenítő (Presenter) fogja össze az alkalmazáson belül az összes komponens működését. A Smalltalk féle Kontroller funkcionalitását az alkalmazási réteg szintjére emelte, feladata a megfelelő Modellek, Szelekciók, Parancsok, Nézetek, Interaktorok létrehozása, az alkalmazás munkafolyamatának irányítása.

A legszembetűnőbb különbség a Taligent féle MVP és az MVC között a Megjelenítőben és az Interaktorokban rejlik. A Presenter egyfajta általános menedzser szerepet tölt be, a felhasználói események elkapásáért ugyanakkor nem a Presenter hanem az Interaktorok a felelősek, így nem szükséges minden egyes Nézet-beli grafikus elemhez (widgethez) saját Presenter, mint ahogy az volt a Smalltalk Kontrollerek esetében. Általában egy adott Nézethez egy Presenter tartozik, bizonyos esetekben egy Presenter akár több Nézetet is kezelhet.

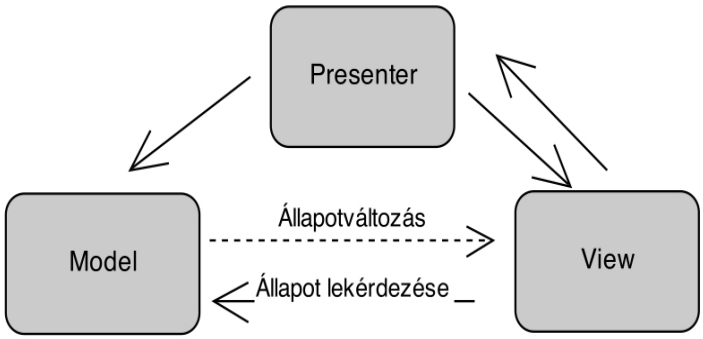
Igazából az Interaktorok hasonlítanak a Smalltalk-80-as Kontrollerekre, ők azok, akik reagálnak a felhasználói eseményekre és meghívják a Modell megfelelő Parancsait és Szelekcióit [42].

Dolphin Smalltalk MVP

A Dolphin Smalltalk csapat egyszerűbbé tette az eredeti MVP mintát abban az értelemben, hogy kihagyta az Interaktor, Parancs, Szelektor elemeket az architektúra definíciójából, sőt a Megjelenítő funkciója is egyszerűbbé vált; egy alrendszer menedzselő komponensből egy mediátorrá vált, mely feladata a Modell frissítése a Nézettől érkező információk alapján. Észrevették, hogy az MVC Kontroller fogalma, mely elsődleges feladata a felhasználói eseményekre történő reagálás volt, már nem igazán felel meg a mai fejlesztői keretrendszereknek, ahol a natív widgetek már közvetlenül képesek ezeket az eseményeket kezelni.

No persze itt azért rögtön meg kell jegyeznünk, hogy azon GUI elemek, widgetek működése - amelyekkel ezek a keretrendszerek alapból rendelkeznek - nyugodtan ábrázolható és leképezhető az eredeti MVC architektúrára, sőt a mai méltán népszerű kliens oldali JavaScript / Ajax keretrendszerek pontosan ezt is teszik.

Ennek megfelelően a Dolphin MVP tervezési mintában (15. ábra) a Nézet az, aki elfogja a felhasználó által generált eseményeket. A Nézet ezután delegálja ezen eseményeket a Megjelenítő részére, aki aztán ennek megfelelően módosítja a Modellt. Így egy fontos különbséget észrevehetünk az MVC és a Dolphin MVP között, nevezetesen azt, hogy ebben az esetben a Megjelenítő (Presenter) elsődleges feladata a Modell módosítása, a felhasználói események elkapása egyfajta melléktermék, másodlagos funkció, hisz ez a feladat elsősorban a Nézetre hárul [43].



15. ábra: A Dolphin Smalltalk MVP

MVVM

Végezetül nem szabad megfeledkeznünk egy, leginkább Microsoft platformú fejlesztők körében használatos tervezési mintáról az MVVM-ről sem (16. ábra).

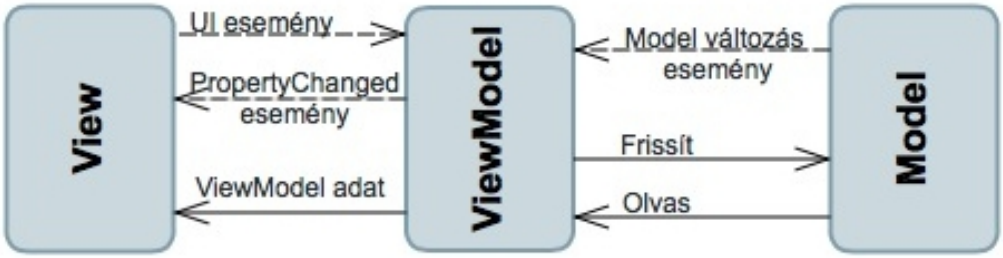
Az MVVM a Martin Fowler-féle Presentation Model [44] Microsoft által implementált változata, népszerűvé és ismertté a Microsoft Presentation Foundation (MPF) és a Silverlight keretrendszerekben történő alkalmazásával vált [45].

Az MVVM fő feladata a felhasználói felület és a mögötte meghúzódó üzleti logika kettéválasztása. Segítségével az alkalmazás egyes komponensei egyszerűbben tesztelhetőek, valamint jóval könnyebben, egymástól függetlenül fejleszthetőek. Az MVVM minta az alábbi elemekből épül fel:

- Modell (Model), az MVC architektúrában megszokott Modell fogalmának felel meg.

- Nézet (View), maga a felhasználói interfész. Információt jelenít meg a felhasználó számára illetve eseményeket indít el a felhasználói interakciók hatására.

- NézetModell (ViewModel), a híd a Nézet és a Modell között. Minden Nézet rendelkezik saját NézetModellel. A NézetModell adatokat kap a Modelltől, majd a Nézet számára megfelelő formátumúvá alakítja. Értesíti a Nézetet, ha a mögöttes adat megváltozott, valamint a felhasználói felületen bekövetkezett események alapján frissíti a Modellbeli adatokat [46].



16. ábra: A Model-View-ViewModell tervezési minta

Bár a felépítés több szempontból hasonlít az MVP architektúrájára, a Microsoft gondolata szerint az MVP inkább a hagyományos szerver által generált weboldalak és a kérés / válasz paradigma megvalósítására alkalmas, míg az MVVM a gazdag kliens alkalmazások megvalósítására optimalizált minta, ahol a kliens-oldali üzleti logika és az alkalmazás állapota a felhasználói és szolgáltatások közötti interakciók által van fenntartva. A minta segítségével könnyen megvalósíthatóak a kétirányú adatkötések, például deklaratív megközelítésű adatkötés használható a Nézet és a NézetModell összekapcsolására, ahelyett hogy programkódokat kellene írnunk a kettő összekötésére. Így a Nézet tulajdonságait a NézetModellhez kapcsolva egyetlen sor kódot sem kell írnunk ahhoz, hogy a NézetModell akkor frissítse a Nézetet amikor szükséges; a kötéssel ez automatikusan megtörténik.

Az MVP Megjelenítő elemétől eltérően a NézetModellnek nincs szüksége a Nézetre mutató hivatkozásra. A Nézet és a NézetModell összekötése lehetővé

teszi, hogy ha NézetModell-beli tulajdonság értéke változik, ezen új érték– az adatkötés miatt - automatikusan továbbítódik a Nézethez. Amikor a felhasználó a Nézeten rákattint egy gombra, egy NézetModell-beli parancs lefut és végrehajtja a kért feladatot. A NézetModell, és sohasem a Nézet az, aki a Modell-beli adatokon módosításokat végez.

Több fejlesztő ugyanakkor megkérdőjelezi a minta, pontosabban a minta gyakorlati alkalmazásának helyességét. A probléma ugyanis az, hogy igen sokan az üzleti és alkalmazáslogikát a NézetModellbe akarják belesűríteni.

A Nézetbe nem tudják, hisz az csak egy egyszerű felhasználói felület. Nem tudja, hogy az adat honnan érkezik, csak annyit tud, hogyan kell azt megjeleníteni.

A Modell tartalmazza magát az adatot, melyből a NézetModell gazdálkodik, oda nyilván megint nem kerülhet az alkalmazáslogika.

Marad tehát a NézetModell, amely alap esetben a Modell-beli adat egy részhalmazát, valamint azokat a parancsokat tartalmazza, amelyekhez a Nézet kötve van. Ebbe beletéve még az alkalmazáslogikát is, a NézetModell hatalmasra duzzad, a hibajavítás, tesztelés rendkívül nehézkessé válik.

Épp ezért indokolt az MVVM modellt kiegészíteni egy Kontroller elemmel, ezt emlegetik a szakmában MVCVM architektúrának.

A Kontroller feladata, hogy az alkalmazáslogikát megvalósítsa, továbbá, hogy az egymástól független komponenseket egy komplett alkalmazássá fogja össze. Ezzel funkcionalitással korábban már találkoztunk, noha ott és akkor más névvel illették. Valóban, a Taligent MVP Megjelenítő komponense épp ezt a funkciót látta el.

Látható tehát, hogy az MVC architektúrát sokan sokféleképpen alakították át, a kor vagy épp saját technológia igényeiknek megfelelően, sőt egyazon tervezési mintát más és másféleképpen értelmeznek a szakemberek. Jól jellemzi a helyzetet Josh Smith [47] találó kijelentése:

„Ha egy szobába leültetsz tíz programtervezőt beszélgetni arról, mi is az a Model-View-Controller tervezési minta, a beszélgetés végeztével tizenkét különböző véleménnyel távozol.”

A saját tervezési minta

A saját modell kidolgozásának ötlete reakció az iparból és a szolgáltatási szektorból (pl. turisztikai webportálok) érkező fejlesztő igényekre, tapasztalatokra. Az új tervezési architektúrával igyekeztem formalizálni az eddigi ad-hoc jellegű megoldásokat, egyben rögtön gyakorlati mintát is kínálok az elméleti kutatás igazolására.

A kiindulási rendszer

A különböző tervezési minták ismertetése során észrevehettük, hogy a klasszikus MVC mintától eltérő MVP vagy MVVM rendszerek mindkét esetben a Kontrollert igyekeztek lecserélni, majd az is világosan kiderült, hogy Kontroller funkció nélkül nem készíthetünk komplex alkalmazást. Számunkra egyébként is szimpatikus az eredeti MVC architektúra, annak ugyan a kissé módosított, Cocoa verziója, melyben a Nézet és a Modell a Kontrolleren keresztül lép kapcsolatba egymással. Választásunkat indokolja továbbá az a tény, hogy mind kliens, mind szerver oldalon az MVC keretrendszerek a legelterjedtebbek. Nézzük hát, hogy hogyan és épülnek fel a gyakorlatban ezek a rendszerek.

A szerver oldali megoldásokkal a fejlesztőknek nagyobb tapasztalatuk van, leginkább ehhez az eszközkészlethez nyúlnak. A Modell-Nézet-Kontroller hármast az alábbi példán keresztül szemléltetjük.

A Modell felelős az adatbázissal történő kapcsolattartásért, ő olvassa ki és írja a táblaadatokat, valamint a Modell tartalmazza azokat a PHP osztályokat, melyeket az adatkezeléshez, adatstruktúra reprezentálásra írtunk. Ilyen osztály lehet pl. egy Szemely, melynek tulajdonságai a nev, cim, email, telefonszam, míg metódusai a getNev(), setNev() stb. Nyilvánvalóan a metódusok megfelelően vannak implementálva annak érdekében, hogy adatbázis művelet előtt például legyen adatvalidálás.

A Kontroller feladata, hogy a Modelltől érkező adatokat előkészítse a Nézet számára vagy épp a Nézeten bekövetkezett felhasználói interakció alapján adatokat kérjen a Modelltől. Ezen interakciók tipikusan HTTP (GET vagy POST) kérések formájában érkeznek. Ahogy korábban jeleztük is, az alkalmazáslogika is a Kontroller modulba kerül, ez már a fogalom kiterjesztése a klasszikus MVC felépítéshez képest.

A Nézet felelős az aktuális HTML tartalom legenerálásáért, megjelenítéséért; PHP környezetben egy nézet tipikusan egy template. A templatek szabványos HTML és ún. template -{tag}- tagokat tartalmaznak. Ezen {tag}-ok lehetnek egyszerű változók, metódusok, iterációk, szelekciók, melyeket a template motor php kóddá fordít, a kód futtatásának (értelmezésének) eredménye pedig bekerül {tag}-ok helyére. Bár natív PHP programozás során is bátran kombinálhatjuk a PHP kódot HTML tagokkal, a template tag-ok átláthatóbbak, rövidebbek és ideális esetben nem függnek a mögöttes nyelvtől.

A template-k használatával jól kezelhető a Nézet és a Modell kettéválasztása, ám mára, a kliens oldali gazdag tartalom miatt egyre nehézkesebb a velük való munka. Míg korábban elegendő volt a template-hez hozzácsatolni egy JavaScript könyvtárat és alkalmazni annak néhány metódusát, ma már igen súlyos kódok terhelik az olykor egyébként is vaskos Nézetet. A kliens oldali fejlesztők munkája megnőtt, saját eszközkészletet, keretrendszert használnak, melyhez nem illeszkedik a PHP rendszer által nyújtott szolgáltatás.

Ezen túlmenően a kliens- és szerver-oldali fejlesztési munka jelen módszerrel nehezen párhuzamosítható, ideális lenne egy olyan architektúra, melyben egymástól függetlenül tud dolgozni a kliens-oldali és a szerver-oldali programozó.

Egy olyan ajánlásra lenne szükség, melyben a szerver oldali programozónak nem kell értenie a JavaScripthez, a kliens oldali fejlesztőnek pedig a PHP-hoz vagy épp a Smarty templatehez.

A Kliens-Szerver MVC integrálása

A kérdés tehát az, hogyan lehet összekapcsolni a JavaScript MVC-t a PHP MVC-vel úgy, hogy közben az összekapcsolt rendszerünk ugyanúgy megfeleljen a Modell-Nézet-Kontroller felépítésnek.

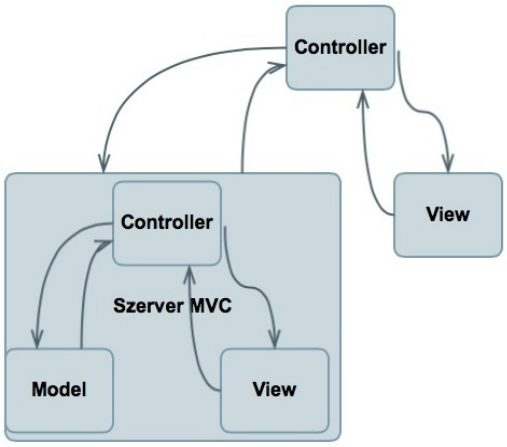
Szerver oldalról megközelítve egyértelműen a Nézet komponens az, ami további szegmentálásra szorul, hisz ennek bonyolultsága az, ami nehezíti a fejlesztési munkát. Lecserélve a Nézetet egy komplett kliens oldali MVC-re a rendszerünk MVC-ből M(MVC)C-é változik.

Amennyiben kliens oldalról vizsgáljuk a helyzetet, az MVC architektúra Modell komponense maga a sima HTML kód, a Nézet a CSS fájl (ahány CSS

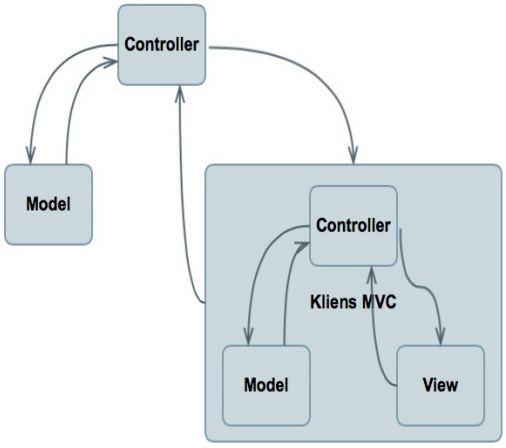
fájl annyi nézet), míg a Kontroller maga a böngésző, illetve a böngésző képességeit kiterjesztő JavaScript programkód. Egy másfajta megközelítésben – különösen ha a kliens-szerver között asszinkron, például AJAX-alapú kommunikáció zajlik-, a Nézet a HTML+CSS+adat kombinációjából megszületett felhasználói felület, a Kontroller szerepét JavaScript osztályok és metódusok tölti be, míg a Modell nem más, mint a webszervertől érkező adat.

Bármelyik szemléletet is tekintjük, kliens oldali megközelítésben a Modell az a komponens, melyen keresztül a rendszerünk illeszthető a szerveri oldali MVC rendszerhez.

Ésszerű gondolatnak tűnik tehát, hogy a kliens oldali MVC a szerver oldalról érkező adatokat tekintse a saját Modellje adatforrásának (17. ábra), míg a szerveri oldali rendszer a kliens oldal kiszolgálását tekintse a Nézet funkcionalitásnak (18. ábra).

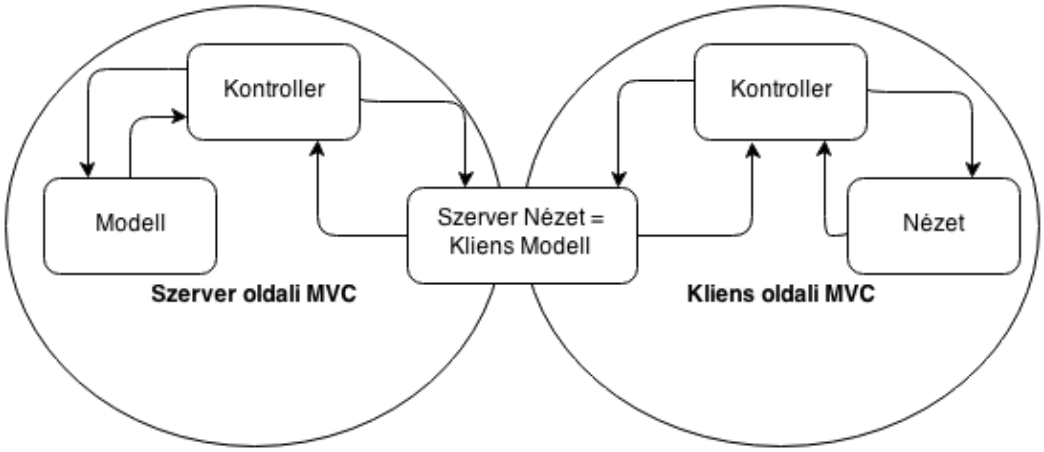


17. ábra: Kliens oldali MVC



18. ábra: Szerver oldali MVC

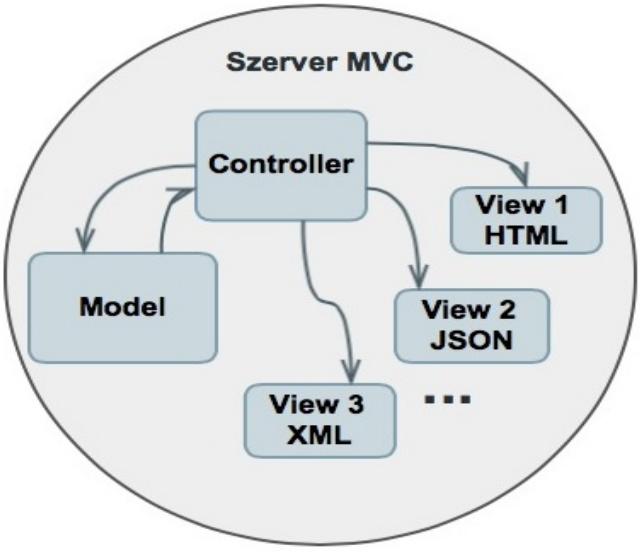
Ehhez ezek szerint arra van szükség, hogy a kliens MVC Modelljét, illetve a szerver MVC Nézetét kidobjuk? Nem, a megoldás célja épp az, hogy érintetlenül hagyjuk a meglévő keretrendszereket, ám megoldjuk a kettejük közötti kapcsolatot. A híd, a megoldás egy közös interfész, ami a kliens oldali Modellt megfelelteti a szerver oldali Nézetnek (19. ábra).



19. ábra: Az Integrált MVC modell

Az újonnan megalkotott architektúra egyben azt is támogatja, hogy egyazon PHP rendszerhez több különböző kliens oldali MVC rendszer csatlakozzon akár úgy is, hogy más-más formátumú adatot kívánnak.

Mivel a tervezési minta lehetővé teszi, sőt támogatja egyazon rendszerhez több Nézet illesztését, ezért a szerver oldali rendszer Nézeteinek számát igényeink szerint bővíthetjük. Ezáltal lehetőséget biztosítunk arra, hogy komplett HTML lapok mellett tetszőleges, a kliens oldali rendszereknek tetsző formátumot szolgáltasson. Ilyen jellemző formátum a JSON vagy az XML (20. ábra).



20. ábra: Szerver MVC - Többféle Nézet

Fontos kérdés, hogy az új összekapcsolt rendszerünk továbbra is tükrözi e az MVC architektúrát? Nos, bármelyik irányból is közelítünk a Cocoa-féle szerkezeti felépítés visszaköszön, gyakorlatilag az egyes rendszerek szempontjából változás nem történt, mindössze a kimenet (Nézet) formátuma, illetve a bemenet (Modell) forrása változott meg.

A közös interfész

A gyakorlati szemlélet megkívánja, hogy rövid példával illusztráljuk, hogyan tud egy tetszőleges JavaScript MVC keretrendszer adatot kérni és a szervertől kapott adatot feldolgozni. Ahhoz, hogy teljes képet kapjunk a közös interfész szerver oldali kódját is ismertetem, a beérkezett kérés feldolgozását és az arra adandó válasz megküldését. A közös formátum ez esetben a JSON lesz, az egyszerűség kedvéért szerver oldalról konstans és nem adatbázisból kinyert adattal válaszolunk.

Kliens oldali szkript

```
function getDomainList() {
    $.ajax({
        type: 'POST',
        url: '../ajax/json_server.php',
        data: 'q=dlist',
        dataType: 'json',
        cache: false,
 //amennyiben sikeresen megérkezett az adat, feldolgozzuk
        success: function(data) {
         var items = [];
   $.each( data, function( key, val )
   {
    items.push( "<a href='http://" + key + "'>" + val +
"</a><br/>" );
   });
 $('#list').append( items.join('') );
  },
  //ha hiba történt, akkor azt jelezzük
  error: function(){
      $("#list").html('JSON adathiba!');
         }
    });
    return false;
}
```

A függvényben szereplő `$.ajax()` metódus felelős az asszinkron HTTP kérés összeállításáért és kezeléséért, ahol a `type` a kérés típusának beállítására szolgál (GET vagy POST), az `url` a kérést fogadó szerver oldali szkript elérési útját adja meg, a `data` a szervernek küldendő adatot jelenti, a `dataType` a szervertől visszaérkező adat típusát jelzi, míg a `cache` paraméter igaz vagy hamis értékével adhatjuk meg, hogy a böngésző a kért oldalt gyorstárazza e be

vagy sem. A `success` kulcsszó utáni függvény abban az esetben fut le, ha a kérés sikeres volt. Esetünkben a szervertől érkező adatokat a *list* nevű konténer elemben jelenítjük meg, az ide vonatkozó HTML forráskód részlet az alábbiak szerint néz ki.

```
<body>
 <script>
  getDomainList();
 </script>
 <div id="list"></div>
</body>
```

A szerver oldali kód

```
<?php
if ( isset($_POST['q']) && ($_POST['q']=="dlist"))
 {
//összeállítjuk a kérésre adott választ
    $domain_list["www.unideb.hu"]= "University of Debrecen";
    $domain_list["www.stanford.edu"]= "Stanford University";

//majd JSON formátumban megküldjük a kliensnek
    echo json_encode($domain_list);
 }
?>
```

Az adatkommunikáció az alábbiak szerint néz ki (S – Szerver, K– Kliens):

A kliens oldal egy asszociatív tömböt ($_POST) állít össze és küld meg a szervernek:

Kmodel → `Array ( [q] => dlist )` →SView

Melyre válaszul a szerver JSON formátumú adatot küld vissza:

SView → `{"www.unideb.hu":"University of Debrecen","www.stanford.edu":"Stanford University"}` → KModel

A fenti JSON formátumot a kapott paraméterei alapján a `json_encode()` PHP függvény állítja össze. Esetünkben ez a paraméter a `$domain_list` asszociatív tömb volt.

Az új rendszertervezési modell megfelelő megoldást kínál a mai modern rendszerek fejlesztéséhez úgy, hogy közben megőrzi a hagyományos értelemben vett MVC paradigmákat. További előnye a megalkotott modellnek az eddigiekhez képest, hogy segítségével egyszerűen integrálhatunk különböző szerver és kliens oldali rendszereket.

Új modell a webrendszerek gyorsabbá tételére

A mai, webbel kapcsolatos kutatások jelentős része foglalkozik az intelligens webrendszerek fejlesztésével, az ajánlói rendszerek fejlesztése különös figyelmet, prioritást élvez. A jelenség magától értetődő, hisz a weben elérhető információ mennyisége folyamatosan nő, a jövőben csak azok a web szolgáltatások maradhatnak talpon, akik személyre szabott tartalmat képesek szolgáltatni látogatóik számára. Ez utóbbi megállapítás ma aktuálisabb mint valaha, ugyanakkor erre már 2001-ben rájöttek a szakemberek, Rossi és társai [48] már ekkor elkezdtek foglalkozni a megszemélyesített webalkalmazásokkal.

Azóta is számos remek cikk foglalkozott már a tartalom-alapú [49], együttműködés-alapú [50] vagy épp a tudásbázis-alapú [51] ajánlói rendszerekkel, ám ma már ez nem elég.

A mobileszközök rohamos terjedése és az Dolgok Internetének [52] megjelenése ma már megköveteli az eszközfüggetlen, reszponzív weboldalak létrehozását.

Problémák

Nem számít, milyen briliáns egy matematikai modell, mennyire hatékony a webrendszerünk mögött meghúzódó mesterséges intelligencia, ha nem tud egységnyi idő alatt megfelelő információt megjeleníteni a weboldalon, elveszítjük a látogatókat. A megdöbbentő, hogy az a bizonyos egységnyi idő

mindössze 3 másodperc.

A StrangeLoop Networks és a PhoCusWright 2010-es kutatási igazolják, hogy 3 másodperc várakozás után az online vásárlók 57 százaléka otthagyja a weboldal. 80 százaléka ezen vásárlóknak többé nem is tér vissza, és több mint a felük ezen negatív tapasztalatát el is meséli ismerőseinek [53].

Lohr 2012-ben, a New York Times-ban megjelent cikke szerint, - mely a Google és a Microsoft mérnökeinek kutatásain alapul - az emberek jóval kevesebbszer látogatják azokat a weblapokat, amelyek több, mint 250 ezredmásodperccel lassabbak versenytársaiknál [54].

Valljuk be, 250 milliszekundum nem sok, ám a kutatások alapján mégis szignifikáns, ha egy weboldal sebességérzetéről beszélünk. Még rosszabb hír, hogy mindössze 1 másodperc késedelem a weboldal betöltésében 7 százalék konverzió vesztést, 11 százalékkal kevesebb oldalmegtekintést és 16 százalékos vásárlói elégedettségcsökkenést eredményez [55]. Az Econsultancy legújabb kutatása szerint a világ online kereskedelme a lassú weboldal-betöltések miatt minden évben 1,73 milliárd font bevételtől esik el [56].

Mindezeken felül egy átlagos felhasználó a ténylegesnél 15 százalékkal lassabbnak érez minden weboldal letöltést. Amikor pedig megosztják másokkal az oldal sebességével kapcsolatos tapasztalataikat, már 35 százalékkal lassabbnak állítják be, mint az ténylegesen volt [57].

A sebesség tehát kardinális probléma, hisz minden intelligens web szolgáltatás, minden ajánlói rendszer arra született, hogy szolgálja az eKereskedelmet, növelje a látogatók számát, növelje a bevételt és a profitot. Természetesen az intelligens webrendszerek fontos szerepet játszanak az oktatás, a kutatás és a tudomány számos más területén, ám mégis, az üzleti világ az, ahol webrendszerünk sikerét pénzben és ezredmásodpercekben mérik.

Épp ezért a weboldalaknak minden eddiginél gyorsabbnak, okosabbaknak kell lenniük, a webfejlesztőknek és teljesítmény optimalizáló szakembereknek le kell szorítaniuk az oldal betöltési időt 3 másodperc alá. Sőt, 2010-től immár azoknak is, akik a Google találati listáján előrébb szeretnének kerülni, ugyanis ettől az évtől kezdve hivatalosan is Google PageRank [58] értékelési paraméter egy weboldal sebessége [59].

Az értekezés ezen részében a Google Developer [60], a Yahoo Yslow [61], valamint Shouders O'Reilly gondozásában megjelent írásain [62] [63] túl alkalmaztam a saját kutatási és fejlesztési tapasztalataim során használt optimalizálási technikákat.

Rendszereztem, kategorizáltam és összegeztem a jelenleg fellelhető és alkalmazható technikákat (Függelék - 1. táblázat), majd tesztelési folyamatok sora után kiválasztottam azokat, melyek a legjobb hatásfokkal bírnak egy weboldal sebességének növelése szempontjából.

Elméleti háttér

A probléma és a megoldás megértéséhez nem kerülhetjük meg a HTTP kommunikáció ismertetését, a protokoll megfelelő mélységű tárgyalása, ismerete szükséges ahhoz, hogy a kutatás alkalmazott módszerei világossá váljanak számunka.

HTTP Request

A HTTP (Hypertext Transfer Protocol) egy, a World Wide Web Consortium (W3C)[9] és az Internet Engineering Task Force (IETF)[10] gondozásában lévő átviteli protokoll elosztott, kollaboratív, hipermédiát használó információs rendszerekhez. A HTTP 1990 óta a World Wide Web (WWW) általános átviteli szabványa, egy kérés-válasz alapú protokoll kliens és szerver között.

Az első verzió a HTTP/0.9[11] természetesen Tim Berners-Lee nevéhez fűződik, majd 1996-ban, immár RFC1945 (Request For Comments) szintre emelve megjelent a HTTP/1.0 verzió[12] . Jelenleg a HTTP/1.1 (RFC2616) verzió az aktuális, mely az eredetileg 1997-ben kiadott RFC2068 '99-es, javított kiadása.

Joggal merül fel a kérdés, hogy az azóta eltelt 14 év alatt nem jelentkezett igény újabb módosításokra? Dehogynem. 2007-ben létrejött a HTTPBIS[13] munkacsoport, mely egyrészt egyértelműsíteni kívánja a jelenleg érvényben lévő 1.1-es verziót, másrészt az új technológiai igényeknek megfelelő HTTP/2.0 bevezetését szorgalmazza.

A munka fontosságát és jelentőségét mutatja, hogy a munkacsoport vezetője

9 http://www.w3.org
10 http://www.ietf.org
11 http://www.w3.org/Protocols/HTTP/AsImplemented.html
12 http://tools.ietf.org/html/rfc1945
13 http://tools.ietf.org/wg/httpbis/

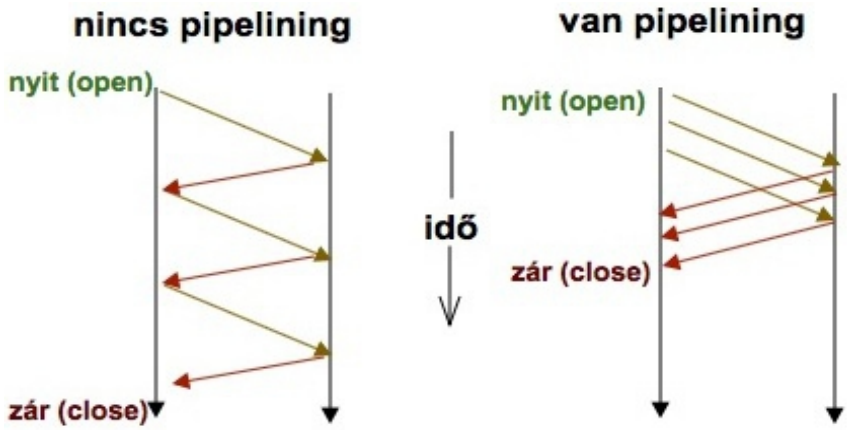
Mark Nottingham, aki korábban a Yahoo!, most az Akami vezető tervezője; a szabvány koordinálását olyan szerzők jegyzik, mint Mike Belshe a Google Chrome korábbi fejlesztő mérnöke és Roberto Peon, a Google mérnöke. Hozzájuk csatlakozva szerkesztőként Martin Thomson, a Microsoft, valamint Alexey Melnikov az Isode részéről. A vitaanyag jelenleg is nyitott, a 2013. októberi, hetes sorszámú draftverzió szerint 2014. április 24-ig várják a szakemberek a korrekciókat, módosító javaslatokat [64].

Miért térnek el egymástól a különböző verziók, miért volt szükség a protokoll újragondolására? A HTTP/1.0 legnagyobb korlátja abban rejlett, hogy a kliens-szerver között felépített TCP kapcsolat kiépítése után mindössze 1 darab kérés mehetett a szerverhez, majd arra szintén csak 1 darab válasz érkezhetett; ezek után a TCP kapcsolat lebomlott. Újabb kérés esetén új TCP kapcsolat, újabb kérés-válasz pár, majd ismét kapcsolatbontás. Ez a 'korabeli' weboldalaknál elfogadható volt, ám ma már, mikor megannyi multimédiás elemet, szkriptet, stíluslapot tartalmaz egy oldal ez nem kielégítő megoldás.

Épp ezért, a HTTP/1.1 bevezette a *pipelining*-ot[14], ami ebben az esetben azt jelenti, hogy egy TCP kapcsolat felépítése után ugyanazon a csatornán több kérés-válasz közlekedhet egymást követve úgy, hogy közben a TCP kapcsolat folyamatosan megmarad. A technológia révén a kliens anélkül tud újabb kéréseket küldeni a szerver felé, hogy megvárná az előző kérésére adott választ, ami nagymértékben gyorsítja a kommunikációt (21. ábra).

Szakemberek ugyanakkor rámutatnak arra is, hogy ez a fejlesztés csak részben oldotta fel a kérések torlódását, hisz egyrészt mind a szervernek, mind a kliensnek támogatnia kell ezt a technológiát – sok böngészőben ez a funkció alapértelmezés szerint ki van kapcsolva [65] - másrészt a szervernek a beérkező kérések sorrendjével megegyező sorrendben kell a válaszokat is szolgáltatnia. Így egy kapcsolatra nézve a sorrend a First-In-First-Out (FIFO) formulát követi. Ennek megfelelően egy-egy robusztusabb méretű válasz – egy nagyobb méretű kép vagy videó - akár súlyos tizedmásodpercekig képes feltartóztatni a mögötte torlódó forgalmat, mely nagymértékben lassítja 1-1 weboldal betöltését.

14 http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1.2.2



21. ábra: Pipelining technológiával és anélkül

Ezen problémát hivatott feloldani a HTTP/2.0, mely - többek között – lehetővé tenné a kérések prioritás szerinti kiszolgálását, azaz a legfontosabb kéréseket előrébb véve nagymértékben javítani lehetne a teljesítményen. A jelenleg is használt HTTP/1.1 üzenet formátumának kialakításánál ugyanis nem a teljesítmény optimalizálás, mint inkább az egyszerű megvalósíthatóság és elérés biztosítása volt az elsődleges cél. A leendő 2.0 szabvány azt is lehetővé teszi, hogy egyazon kapcsolaton asszinkron módon, a kérésekre adott válaszokat nem bevárva, azok sorrendjét nem FIFO szerint rendezve, egymástól független kérés-válasz párokat lehessen elhelyezni.

A végeredmény tehát egy jóval Internet- és hálózat-barátabb protokoll, mely használatával a korábbiakhoz képest lényegesen kevesebb konkurens TCP kapcsolatot kell fenntartani egy-egy kliens-szerver kommunikáció során.

Az elméleti bevezető és technológiai háttér bemutatása után nézzük meg, hogyan is működik a gyakorlatban a HTTP.

HTTP a gyakorlatban

Ahogy korábban is említettem, az első lépés az, hogy a kliens kérést (request) intéz a szerver felé, az alábbi formátum szerint:

```
GET /news/ HTTP/1.1
```

A sor értelmezése igen egyszerű, metódus-erőforrás-protokollverzió hármasként értelmezve, a metódus a `GET`, az erőforrás a `/news/` mappa, míg a verziószám az `1.1`.

Ezt a sort tetszőleges számú HEADER sor követheti

```
Host: www.nagyzsolt.hu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0
```

A fenti példában

- a `Host` jelzi az lekérendő erőforrás Internet host nevét vagy IP címét és portszámát,
- a `User-Agent` sor a kliens által használt webböngésző típusát adja vissza,
- a z `Accept` -tel jelzi a kliens a szervernek, hogy válaszként milyen típusú médiát hajlandó elfogadni,
- az `Accept-Language` értelemszerűen a válasz preferált nyelvét jelzi
- a z `Accept-Encoding` a válasz - kliens által - elfogadható kódolását jelzi a szerver számára (ebben az esetben a kliens képes feldolgozni akár `gzip`-pel, akár `deflate`-tel tömörített válaszokat is)
- a `Connection` értékkel jelzi a kliens a szerver számára, hogy a kérése megválaszolása után szeretné e lebontani vagy fenntartani a TCP kapcsolatot. Ebben az esetben a `keep-alive` a fenntartást jelzi, ellenkező esetben a `close` értéket küldené a kliens a szervernek.
- Végezetül a `Cache-Control` arra hivatott, hogy szabályozza a kommunikáció alatt megvalósuló gyorstárazást. A `max-age` jelzi, hogy a kliens az itt paraméterként (másodpercben) megadott értéktől 'öregebb'

választ nem hajlandó elfogadni. A `max-age=0` ebben az esetben azt jelenti, hogy nincs gyorstárazás, a szervernek mindig friss tartalmat kell szolgáltatnia válaszként.

A szabvány az itt felsoroltakon túl számos egyéb HEADER paramétert is definiál, a kutatás fázisában elengedőek ezek az információk.

Miután a kérés elment a szerver felé, arra válasz érkezik, melynek szintén szabványos formátuma van, ez a következő:

A HTTP válasz első sora a státuszsor, mely szintén egy hármas: a *verzió – státuszkód - szöveges indoklás* formátumot követi, az egyes elemeket 1-1 szóköz választja el egymástól.

Ilyen például a

```
HTTP/1.1 200 OK
```

ahol a verziószám az `1.1`, a státuszkód a `200`, míg az indoklás, magyarázat az `OK.` vagy a

```
HTTP/1.1 404 Not Found
```

ahol a 404-es hiba jelzi, hogy a kliens által kért erőforrást a szerver nem találta meg.

A szabvány 40 különböző státuszkódot definiál, ám lehetővé teszi, hogy ez a kódlista bővíthető legyen. 5 különböző kategóriába soroljuk a státuszkódokat, a 3 jegyű kód első számjegye határozza meg a kategóriákat. Ennek megfelelően

- 1xx: Informatív – a kérés megérkezett, a feldolgozás folytatódik
- 2xx: Siker – a kérés sikeresen megérkezett, értelmezve és elfogadva
- 3xx: Átirányítás – a kérés teljesítéséhez további műveletre van szükség
- 4xx: Kliens hiba – a kérés szintaktikai hibát tartalmaz vagy nem teljesíthető
- 5xx: Szerver hiba – a szerver nem tudja teljesíteni az egyébként érvényes kérést

A státuszsor után – hasonlóan a kérések (request-ek) felépítéséhez – a HEADER sorok következnek.

```
Date: Sun, 27 Oct 2013 09:20:52 GMT
Server: Apache
X-Powered-By: PHP/5.3.10-1ubuntu3.4
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2788
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

- a `Date` a szerver által megküldött válasz dátumát tartalmazza,
- a `Server`, a szerveren futó kiszolgáló szoftvert nevesíti
- a z `X-Powered-By` egy nem szabványos paraméter, a szerver oldali webalkalmazás nyelvét és verziószámát jelöli, esetünkben ez a PHP 5.3.10
- a `Vary` mezőnek akkor van kiemelt jelentősége, amikor a kliens és a szerver közé beékelődik egy közbülső szerverszolgáltatás, aki például a cache-elést végzi. Ha egy alapvetően ugyanolyan tartalom – értsd ez alatt azt, hogy ugyanaz az erőforrás neve - több változatban is elérhető, legyen szó eltérő nyelvről, karakterkódolásról, tömörített vagy nem tömörített verziójáról, úgy megkísérelhetjük kiszolgálóoldalon eldönteni, melyik változat lehet a legmegfelelőbb az ügyfélnek. A döntésben a `Vary` mező értéke segít, jelen esetben ez az érték az `Accept-Encoding`.
- A `Content-Encoding` mező jelző, hogy a válasz milyen módszerrel van tömörítve,
- a `Content-length` paraméter értéke adja meg, hogy a válasz törzsének mekkora a mérete (bájtban)
- a `Keep-Alive` mező a csomópontok közötti perzisztens kapcsolatról ad információt. A `timeout` paraméter jelzi másodpercben, hogy üresjárat esetén meddig marad még fenn a kapcsolat, míg a `max` paraméter az egyidejű perzisztens kapcsolatok maximális számát határozza meg [66].
- a `Connection` ugyanazt a funkciót látja el, mint HTTP kérés esetén
- a `Content-type` pedig a válasz média típusát és karakterkódolását adja meg. A mi példánkban UTF-8 kódolású html tartalmat szolgáltattunk a kliens felé.

Ezeket az információkat normál körülmények között a böngésző (kliens) programok nem jelenítik meg a felhasználó számára, sőt még a webfejlesztési munkafolyamat során is csak a html forráskódig ásunk le leginkább, ám ha szükséges, számos eszköz létezik a még mélyebb szintű elemzésre a HTTP kérések és válaszok felderítésére.

Talán az egyik legegyszerűbb és legnépszerűbb a nyílt forráskódú Firefox kiegészítő, a Firebug[15], ugyanakkor a Google Chrome böngészőt kedvelők számára elérhető a Chrome HTTP Monitor vagy épp a számos más monitoring funkcióra is alkalmas WireShark[16].

A www.nagyzsolt.hu/news kutatási célra kialakított weboldal HTTP kérés-válasz párosa az alábbi.

Kérés (request):

```
GET /news/ HTTP/1.1
Host: www.nagyzsolt.hu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0
```

és az arra adott válasz, a response:

```
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2013 09:20:52 GMT
Server: Apache
X-Powered-By: PHP/5.3.10-1ubuntu3.4
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2788
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Ezzel gyakorlatilag lekértük és megkaptuk a nyitóoldal html kódját.

15 http://getfirebug.com
16 http://www.wireshark.org

Ugyanakkor maga az oldal számos egyéb erőforrást – képet, szkriptet, stíluslapot – is tartalmaz, ezért ezeket is le kell kérnie a böngészőnek, annak érdekében, hogy a weblapot teljes terjedelmében meg tudja jeleníteni.

A weboldal stíluslapjának lekérése az alábbi HTTP kérés-válasz párossal történik:

kérés:

```
GET /news/css/style.css HTTP/1.1
Host: www.nagyzsolt.hu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.nagyzsolt.hu/news/
Connection: keep-alive
If-Modified-Since: Tue, 08 Oct 2013 17:12:49 GMT
If-None-Match: "a41dd-1fce-4e83de0cca640"
Cache-Control: max-age=0
```

válasz:

```
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2013 09:55:51 GMT
Server: Apache
Last-Modified: Sun, 27 Oct 2013 09:49:02 GMT
Etag: "a41fe-1fac-4e9b5e4b0ff80"
Accept-Ranges: bytes
Content-Length: 8108
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/css
```

A fenti példa a gyorstárazás (cache) nélküli kommunikációt mutatja be. Ugyanakkor – ahogy a következő fejezetben is látni fogjuk - ha bekapcsoljuk szerver oldalon az erőforrások lejárati idejének jelzését az javítja a weboldal megjelenési teljesítményét. Ez esetben a szerver által adott válasz a következőképpen néz ki:

Válasz (response):

```
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2013 09:54:40 GMT
Server: Apache
Last-Modified: Sun, 27 Oct 2013 09:49:02 GMT
Etag: "a41fe-1fac-4e9b5e4b0ff80"
Accept-Ranges: bytes
Content-Length: 8108
Cache-Control: max-age=2592000
Expires: Tue, 26 Nov 2013 09:54:40 GMT
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/css
```

Látható, hogy új elemként megjelent az `Expires` tag, mely értéke egy dátum (RFC 1123 által meghatározott formátumban). Ez a dátum jelzi a kliens felé, hogy az adott dokumentumot meddig kell gyorstáraznia, azaz meddig tekinti úgy az információ szolgáltató (a szerver), hogy az adott erőforrás változatlan, a kliensnek nem szükséges újra és újra letöltenie, eltárolhatja magának a megadott ideig. A szabály úgy szól, hogy ha a HEADER-ben megadott `Date` mező értéke egyenlő vagy későbbi, mint az `Expires` mező értéke, akkor a kliens nem cache-elheti tovább az adott erőforrást, újból le kell kérnie a szervertől.

Fontos megjegyezni, hogy ha `Expires` és `Cache-Control max-age` mezőt is tartalmaz a válasz, akkor a `max-age` felülbírálja az `Expires`-ben megadott értéket.

Érdemes felhívni a figyelmet arra is, hogy az `Expires` mező nem alkalmas arra, hogy kényszerítse a klienst a böngészőablak frissítésére vagy az erőforrás újratöltésére. Mindössze a gyorstárazási mechanizmusban van szerepe, mely abból áll, hogy az adott erőforrás minden egyes lekérdezése előtt meg kell vizsgálnia annak lejárati státuszát.

Felmerül a kérdés, hogy milyen tartalmakat érdemes betáraznia a kliensnek és mely tartalmak azok, melyeket kifejezetten hiba lenne. A saját kutatás-fejlesztési, valamint a szakirodalmi tapasztalatok ebben a témában teljes mértékben összhangban vannak; egy weboldalon található képek, multimédiás tartalmak, stíluslapok és szkriptfájlok alkotják azt a csoportot, melyek nem változnak gyakran, míg a html tartalmak azok, melyeket szükségszerű minden

alkalommal letöltenünk. Miért van ez így, a html lapokat miért ne cache-elhetnénk?

Manapság, amikor a dinamikus weboldalak korszakát éljük, 1-1 komolyabb webportál tartalma akár percenként is frissül, de még egy egyszerűbb céges weblap is hetente változtatja tartalmát attól függően, hogy milyen új akciókat, szolgáltatásokat kínál ügyfeleinek. Ehhez vegyük még hozzá azt a tényt, hogy a html forráskód mérete eltörpül egy-egy kép, hang vagy videofájl méretéhez képest, továbbá olyan funkcióknál – mint például egy keresés eredménye, egy találati lista, egy lekérdezés, egy riport – amelyek valós időben, a felhasználó aktuális preferenciájának és általa megadott paramétereinek függvényében generálják le a tartalmilag mindig más és más html oldalt, hiba is lenne eltárolni egy régi oldalt, hisz az már nem aktuális tartalom.

Ugyanakkor egy weboldalra feltöltött *parlament.jpg* tartalma két nap múlva is ugyanazt a bitkombinációt fogja tartalmazni, mint most; a Parlamentről készült képet. Persze a webfejlesztés során előfordul, hogy ezeket a tartalmakat (kép, stíluslap, szkript) is optimalizálják, módosítják a programozók, így a fejlesztés ideje alatt pontosan ezért nincs gyorstárazás, vagy épp minden egyes oldalmegtekintés előtt kiürítik a böngésző gyorstárát. Ezáltal minden egyes alkalommal a legfrissebb, aktuális tartalom töltődik le a webszerverről. Tapasztalatlanabb weblapkészítők - elfelejtve ezt az apró, de fontos tényt - hosszú perceken, órákon át böngészik a forráskódot, miszerint minden rendben, ám mégsem az a tartalom jelenik meg, mégsem az a szkript fut le a böngészőben, amit ők megírtak.

A HTTP kommunikáció ismeretének birtokában most már nekikezdhetünk az új módszer ismertetésének, mely első lépése a tesztelési környezet kialakítása.

Tesztelési környezet

Ahhoz, hogy a javasolt módszerek hatásfokát megfelelően mérni lehessen, egy tesztelési környezetet szükséges felállítanunk, melyben az eredeti weblap majd az optimalizált weblap sebessége, ugyanazon módszerekkel mérhető. Számos ingyenes és hasznos sebességmérő rendszer található a piacon, a Sixrevisions.com össze is gyűjtötte a legjobb 20 ilyen eszközt [67], a kutatási eredmények mérésére ezek közül ötöt választottam ki, ezek név szerint: a

Google PageSpeed Insights[17], a Yahoo Yslow[18], az AOL WebPageTest[19], a GTMetrix[20] és a Pingdom[21].

A fentiek közül a két meghatározó rendszer a PageSpeed és a Yslow, a többiek javarészt erre a két rendszerre, illetve a Google és a Yahoo által javasolt elvekre épülnek. A WebPageTest és a Pingdom inkább a Google szabályait követik, míg a GTMetrix kombinálja a Yslow és a PageSpeed javaslatait. Ennek megfelelően a leghatékonyabb elemző rendszernek a GTMetrix bizonyult, mely amellett, hogy egy rendszeren belül a legtöbb szabályt képes kezelni, jól használható grafikus felülettel és részletes statisztika-készítési funkcióval is bír.

Előkészületek

Ahhoz, hogy valós képet kapjunk arról, hogyan is állnak jelenleg a világ leglátogatottabb webportáljai sebesség szempontjából, megvizsgáltam az 5 leglátogatottabb weblapot az alábbi kategóriákban:

- a világ első 5 leglátogatottabb weboldala,
- a világ első 5 leglátogatottabb turisztikai oldala, valamint
- Magyarország első 5 leglátogatottabb turisztikai oldala

A választásom az Alexa 2013 júniusi rangsora[22] alapján történt.

17 https://developers.google.com/speed/pagespeed/
18 http://developer.yahoo.com/yslow/
19 http://www.webpagetest.org
20 http://www.gtmetrix.com
21 http://tools.pingdom.com/fpt/
22 http://www.alexa.com/topsites

Látogatók száma szerint	PageSpeed	Yslow	Átlag
Top 5 USA weboldal			
facebook.com	98	97	97,5
google.com	98	94	96
youtube.com	95	86	90,5
yahoo.com	89	77	83
amazon.com	94	82	88
Átlag	**94,8**	**87,2**	**91**
Top 5 Utazási oldal			
booking.com	93	82	87,5
tripadvisor.com	94	92	93
xe.com	89	73	81
expedia.com	81	60	70,5
priceline.com	88	78	83
Átlag	**89**	**77**	**83**
Top 5 magyar utazási oldal			
itthon.hu*	90	78	84
iranymagyarorszag.hu	83	77	80
limba.com	87	83	85
utisugo.hu	86	69	77,5
szallasinfo.hu	76	72	74
Átlag	**84,4**	**75,8**	**80,1**

*az első helyezett szallas.hu oldalt technikai okok miatt – az analizáló eszköz nem volt képes elemezni - lecseréltük. Az itthon.hu a Magyar Turizmus Zrt. által üzemeltetett weboldal.

1. táblázat: Látogatott weboldalak statisztikái

A hatodikként vizsgált magyar utazási portál a BelfoldiSzallasok.hu[23] – ami a magyar utazási oldalak közül a 12-ik – lesz az optimalizálási kutatás tárgya.

A különböző kategóriákat egy közös listába összegezve észrevehetjük, hogy az említett oldal jelen állapotában mindössze a szerény tizenkettedik helyet éri el a tizenhatból.

23 http://www.belfoldiszallasok.hu

#	Összesített – előtte	PageSpeed	Yslow	Átlag
1	facebook.com	98	97	97,5
2	google.com	98	94	96
3	tripadvisor.com	94	92	93
4	youtube.com	95	86	90,5
5	amazon.com	94	82	88
6	booking.com	93	82	87,5
7	limba.com	87	83	85
8	itthon.hu	90	78	84
9	yahoo.com	89	77	83
10	priceline.com	88	78	83
11	xe.com	89	73	81
12	*belfoldiszallasok.hu*	*86*	*76*	*81*
13	iranymagyarorszag.hu	83	77	80
14	utisugo.hu	86	69	77,5
15	szallasinfo.hu	76	72	74
16	expedia.com	81	60	70,5

2. táblázat: Összesített eredmény az optimalizálás előtt

Optimalizálás

A GTMetrix javaslatai és a kutatás során hatékonynak bizonyuló egyéb módszereket sorra véve elvégeztem az optimalizálási tevékenységeket. Az összehasonlíthatóság érdekében az eredeti weboldalt változatlanul hagytam, míg létrehoztam egy alkönyvtárat, mely az optimalizált tartalmat szolgáltatja. Az alábbiakban ismertetem azt a tíz optimalizálási módszert, melyek a leghatékonyabbnak bizonyultak.

Képek fizikai méretének optimalizálása

Egy-egy weboldal – így az általunk vizsgált turisztikai webportál – méretének jelentős részét teszik ki a weblapon található képek. Épp ezért kulcsfontosságú, hogy kiemelt figyelmet fordítsunk a képek méretének optimalizálásra. Az elmúlt 12 év webfejlesztési tapasztalata, - melyet az elmúlt 5 év kutatómunkája és elemzése csak megerősített - azt mutatja, hogy a weblapok tulajdonosai rendszeresen elkövetik azt a hibát, hogy nagyméretű fotót, képet töltenek fel a webszerverre, melyet aztán HTML kód segítségével kliens oldalon igazítanak méretre. Bár a végeredmény kinézetre ugyanaz, ez

több szempontból is lassítja az oldal megjelenítését.

Az elsődleges probléma, hogy a teljes méretű kép utazik a hálózaton a szervertől a kliens felé. Ma már a legolcsóbb digitális fényképezőgép[24] is képes 14 megapixeles képeket készíteni, mely 4000x3000 pixeles felbontást és minimum 1,3 MB JPG formátumú méretet jelent[25]. Ha ezt a képet változtatás nélkül feltöltjük a weboldalunkra, akkor csak ez az egy kép a kliens gépek számára egyenként 1,3 MB adatforgalmat generál. Ugyanakkor jelenleg az átlagos képernyőfelbontás a StatCounter legfrissebb adatai szerint mind Magyarországon, mind világszerte 1366x768 pixel [68]. Ám még ha a full HD szabvány felbontást is vesszük alapul, az is csak 1920x1080, azaz 2,1 megapixel ami mindössze 368 KB.

Tovább árnyalja a dolgot, hogy egy weblapon – a képgalériákat, képnézegetőket leszámítva - ritkán helyezünk el akkor méretű képet, hogy az beborítsa a teljes képernyőt. A Google több milliárd weboldalt felölelő 2010-es statisztikája alapján egy weblapon átlagosan 27,58 db kép található [69].

Ez a szám természetesen tartalmazza a pár kilobyte-os kis ikonokat, elválasztó vonalakat, eltartó elemeket is; ha a kutatás tárgyát vesszük alapul, ez az arány 45%. A webportálunk 40 képi eleméből 18 tartozik ebbe a kategóriába. A Google statisztikáját véve alapul és kissé átbillentve a mérleget a kisképek javára – legyen 60% -, elmondhatjuk, hogy egy átlagos weboldal 11 olyan képet tartalmaz, melynél jelentős méretcsökkenést érünk el, ha a webszerverre való feltöltés előtt átméretezzük.

Így a kliens oldalon a böngészőnek egyrészt nem kell hatalmas méretű képeket letöltenie, másrészt nem kell utána a felhasználói felülethez igazítva kicsinyítenie azokat. Természetesen normál körülmények között a HTML kódban megadott kép szélesség és magasság paraméterek még gyorsítani is tudják a weboldal megjelenítését.

Képek dimenzióinak megadása

A böngészőben megjelenítendő képek szélességének és magasságának megadása gyorsabb felhasználói felület-renderelést eredményez. Amikor ugyanis a böngésző megjelenít egy weboldalt, képesnek kell lennie arra, hogy az áthelyezhető elemek – mint például a képek – köré generálja a tartalmat.

Már a képek letöltése előtt megkezdhető a renderelés, ha tudjuk a letöltendő

24 http://tinyurl.com/p9dcbx

25 http://web.forret.com/tools/megapixel.asp?title=12+Megapixel+camera&width=4000&height=3000

képek dimenzióit (szélesség és magasság).

Ha azonban a dimenzióérték nincs specifikálva a HTML dokumentumban vagy a megadott értékek nem egyeznek a letöltött kép méreteivel, a böngészőnek a képek letöltése után még egyszer újra kell rajzolnia a felhasználói felületet. Ez felesleges munka és időveszteség, ennek elkerülése érdekében mindig adjuk meg a képek méreteit vagy az `<img>` tagban vagy CSS segítségével.

Böngésző gyorstárazás (cache) alkalmazása

A weboldalak mind grafikailag, mind tartalmilag egyre gazdagodnak, ami azt jelenti, hogy még több szkriptet, stíluslapot, képet, flash és egyéb médiát tartalmaznak. Azt gondolhatnánk, hogy ezzel párhuzamosan az Internetes sávszélesség is folyamatosan emelkedik, így képes kiszolgálni az új igényeket is. Ez részben igaz.

Míg 2008-ban 2.5 Mbps volt Magyarországon egy végfelhasználó átlagos sávszélesség kapacitása, addig 2013 első negyedévében ez az érték már több mint a duplájára, 6.6 Mbps-ra emelkedett. Ez az érték akkor is jelentős, ha azt csak az előző év, 2012 első negyedévéhez viszonyítjuk; 12%-os emelkedést jelent [70].

Ám ezzel egy időben az Internetet használók száma is emelkedett. A KSH[26] adatai szerint 2012-ben a magyar felhasználók a lakosság 74%-át tették ki [71], ez az érték 2008-ban még csak 58,7% volt [72], míg 2010-ben is csak 62% [73]. Érezhető hát az utóbbi évek dinamikus fejlődése, látható, hogy a sávszélesség növekedése szerény vigasz; a webszervereknek nagyobb sávszélességen ugyan, ám minden eddiginél több felhasználót kell kiszolgálniuk.

Így érthető az az igény, javaslat, mi szerint az egyszer már kliens oldalra letöltött tartalmat még egyszer ne utaztassuk meg a világhálón, a böngésző tárazza azt be. Ebből kifolyólag nagymértékben csökkenthető úgy a HTTP kérések mérete mint a annak száma, mely végeredményben gyorsabb weboldal betöltődést eredményez.

A webszerver egy `Expires` nevű fejlécet használ a HTTP válaszában annak jelzésére, hogy közölje a kliens oldallal, egy-egy komponens mennyi ideig tárazható be.

Amennyiben egy statikus erőforrás HTTP fejléc részében beállítjuk a lejárati időt vagy a maximális életkort, ezzel utasítani tudjuk a böngészőt arra, hogy a

26 Központi Statisztikai Hivatal - http://www.ksh.hu

korábban már letöltött erőforrásokat ne a hálózatról, hanem a lokális merevlemezről töltse be. Tapasztalatok alapján az Expires értéket legalább 1 hónapra, maximum 1 évre érdemes beállítanunk. A Google az `Expires` tagot preferálja a `Cache-Control: max-age` -el szemben, lévén az előző szélesebb körben támogatott.

Az Expires fejléceket leginkább képeknél használjuk, ám a szkriptek, stíluslapok és Flash komponensek esetében ugyanolyan jó hatásfokkal bírnak. A lejárati időket a szerver oldalon elhelyezett *.htaccess* fájlban tudjuk beállítani. A GTMetrix ajánlása alapján készítettünk egy mintát, mely a leggyakrabban használt képformátumokra (jpg, png, gif és ico) valamint a stíluslapokra és JavaScriptekre ad meg lejárati időt:

```
<IfModule mod_expires.c>
 #Lejárati idő engedélyezése
 ExpiresActive On
 #Az alapértelmezett direktíva
 ExpiresDefault "access plus 1 month"
 #favicon lejárati ideje
 ExpiresByType image/x-icon "access plus 1  year"
 #Képek lejárati ideje
 ExpiresByType image/gif "access plus 1 month"
 ExpiresByType image/png "access plus 1 month"
 ExpiresByType image/jpg "access plus 1 month"
 ExpiresByType image/jpeg "access plus 1 month"
 #CSS
 ExpiresByType text/css "access 1 month"
 #és végül a JavaScript
 ExpiresByType application/javascript "access plus 1 year"
</IfModule>
```

A kódból egyértelműen kiolvasható, hogy a képekre az aktuális hozzáférési időtől – azaz amikor a böngészőnk először letöltötte az adott weboldalt - számított 1 hónapot, míg a favicon és a JavaScript esetében a letöltéstől számított 1 évet adtunk meg lejárati időnek. Ahogy a fenti kódrészlet első sorában láthatjuk, ez a technika feltételezi, hogy a webszerveren a *mod_expires* modul fel van konfigurálva, valamint hogy a *.htaccess* fájlunkat az Apache webszerver képes feldolgozni.

Képek kombinálása CSS Sprite-ok segítségével

Igen jó optimalizálási technika, ha a képeinket CSS sprite-ok segítségével a lehető legkevesebb számú fájlba kombináljuk, hisz csökken a HTTP kérések száma, valamint a letöltött adat mennyisége és a letöltésre fordított idő. Egy olyan weboldal, amely rengeteg képet tartalmaz, a képek néhány fájlba történő összefűzésével nagymértékben csökkenteni tudja a tartalom letöltési idejét. Mit jelent a CSS sprite kifejezés?

Alapvetően arról van szó, hogy egy szoftver segítségével egy nagy felületre egymás mellé és alá másoljuk a képeket, melyet miután elmentünk, CSS segítségével dolgozunk fel kliens oldalon. A CSS segítségével a létrejött egyetlen nagy képünk bármelyik részére tudunk pozicionálni, így mindig a kívánt képrészletet mutatva a weboldal megfelelő helyén. Számos ingyenes sprite szolgáltatást igénybe vehetünk, ilyen a CssSprites[27], a SpriteMe[28] vagy a SpritePad[29]. A SpritePad ingyenes, szép és könnyen használható eszköz, mi az optimalizálás során ezt használtuk. A képek egyszerűen szerkeszthetők, az összerakásuk után letölthetjük mind a létrejött CSS, mind a végeredmény PNG képfájlt.

CSS fájlok minimalizálása

A CSS fájlok méretének csökkentésének eredményeképp a kisebb fájlméret gyorsabb letöltést, feldolgozást és futtatást jelent. Ugyanilyen eredménnyel jár a JavaScript kódok méretének csökkentése is, ráadásul mindkét tömörítési eljárásra léteznek ingyenes eszközök. Ilyen a YUI Compressor[30] vagy a QTMatrix saját beépített eszköze.

A minimalizálás gyakorlatilag annyit tesz, hogy a felesleges szóközöket, sortöréseket eltávolítjuk a stíluslap vagy a szkript kódjából, ezáltal kisebb fájlméretet kapunk. Ez persze később az olvashatóság rovására megy, így amennyiben a jövőben módosítani akarjuk a szkriptünket vagy stíluslapunkat, úgy érdemes megtartani a minimalizálás előtti verziót és azt szerkeszteni, vagy - már minimalizált stíluslap esetében - erre alkalmas célprogrammal visszaállítani.

27 http://www.csssprites.com
28 http://www.spriteme.org
29 http://wearekiss.com/spritepad
30 http://refresh-sf-com/yui

Külső JavaScript fájlok egyesítése, minimalizálása

A külső (.js) JavaScript fájlok egyesítésével hatékonyan gyorsíthatjuk weboldalunk betöltési sebességét, hisz minimalizálhatjuk a többi erőforrás letöltésének késleltetését. Több különálló JavaScript fájl többszöri HTTP kérést kíván és minduntalan megakasztja a weboldal betöltését.

Egy jó webprogramozó természetesen igyekszik modulárisan, újra felhasználható módon felépíteni az általa készített weboldalt, ám amíg ez ideális is követendő fejlesztési metodika, addig arra is tekintettel kell lennünk, hogy a HTML dokumentumba apránként történő modulimportálás drasztikusan képes növelni az oldal betöltési idejét. Legelőször is egy kliens, aki még sosem járt az adott weboldalon – így üres gyorsítótárral rendelkezik róla – kénytelen minden egyes erőforrás megszerzésére külön-külön HTTP kérést küldeni, amely értelemszerűen időbe telik. Másodsorban fontos tudnunk azt is, hogy a legtöbb böngésző mindaddig felfüggeszti a weboldal további részének letöltését, míg egy JavaScript fájl teljesen le nem töltődött és feldolgozásra nem került. Ez minden egyes szkriptfájl esetén további értékes ezredmásodperceket lop el abból a mágikus 3 másodpercből, amennyi idő alatt egy weboldalnak be kell töltődnie.

Szintén hasznos és követendő technika - ahogy azt már láthattuk a CSS fájlok esetében – a JavaScript fájlok méretének csökkentése, minimalizálása.

Egy összetettebb szolgáltatásokat kínáló weboldal esetén a fejlesztők előre gyártott plugin gyűjteményekkel dolgoznak; azok kiegészülve a saját szkriptekkel igen könnyen elérik a 2-3 ezer soros terjedelmet. Egy, a kutatómunka során használt ilyen fájl mérete 172 KB volt, míg a minimalizálás után lecsökkent 111 KB-ra, azaz az eredeti méret 65%-ra sikerült redukálni a fájl méretét.

A minimalizálás itt nem fizikai tömörítést jelent, csupán arról van szó, hogy szépen strukturált programkódból a minimalizálás során kikerülnek a felesleges szóközök, bekezdések, sortörések. Bár ezáltal a programozó számára az új kód igen nehezen olvasható, a kisebb fájlméret gyorsabb letöltést eredményez, sőt a további, tényleges tömörítési eljárás (gzip vagy deflate) még hatékonyabb egy ilyen optimalizálás után.

Számos ingyenes eszköz létezik a fájlok minimalizálására, így bátran használható erre a célra a JSCompress[31], JSMini[32] vagy a korábban már említett YUI Compressor. Természetesen előfordul, hogy egy későbbi időpontban

31 http://www.jscompress.com
32 http://www.jsmini.com

módosítanunk kell akár a CSS, akár a JavaScript állományainkon. A minimalizált verzióban hibát keresni, vagy módosítani igen nehézkes, ezért az eredeti, strukturált szerkezetet érdemes szerkeszteni.

Erre két megoldás létezik; elsősorban eltároljuk tömörítés előtt az eredeti verzió, ha később módosítanunk kell, azt vesszük elő. Másodsorban, ha az eredeti verzió már nincs meg, - mert épp egy több éve működő, esetleg nem is általunk készített webportálon kell módosításokat végrehajtanunk – akkor a deminimalizálás folyamata lesz segítségünkre.

Erre is léteznek szabadon elérhető online eszközök, a munkánk során a JSBeautifier[33], a ProCSSor[34] és a CSS Beautify[35] szolgáltatásait használtuk.

A Google a JavaScript fájlok esetében is tesz ajánlást a fájlméretre vonatkozóan; már 4096 bájt felett érdemes minimalizálni a szkripteket.

Content Delivery Network (CDN) használata

A Yahoo Yslow rendszeresen ajánlja, javasolja a CDN használatát. A CDN, azaz a tartalomtovábbító hálózat olyan webszerverek összefoglaló neve, melyek arra hivatottak, hogy a felhasználók számára minél hatékonyabban szolgáltassák a különböző webtartalmakat. A szerverek differenciált földrajzi elhelyezkedése lehetővé teszi, hogy a felhasználóhoz legközelebbi szerver szolgáltassa számára a tartalmat, ezzel gyorsabb oldalmegjelenítést érve el. A közelség mérőszáma lehet például a válaszidő vagy a hálózati csomópontok száma. A leggyorsabb válaszidejű vagy a klienstől legkevesebb csomópontra lévő webszerver lesz a CDN azon tagja, mely az adott felhasználót kiszolgálja. A tartalomtovábbító hálózatok jellemzően fizetős szolgáltatások, ám a Techyfuzz [74] ajánlása öt ingyenes CDN szolgáltatást is tartalmaz.

Az optimalizálási procedúra során kettőt ki is próbáltam belőle, ám végül nem alkalmaztam. Az indok egyszerű: a kutatási fázis során igyekeztem olyan optimalizálási technikákat alkalmazni, melyek nem igényelnek jelentős módosítást szerver oldalon. A CDN használata ugyanakkor megköveteli a web domainünk DNS rekordjainak módosítását is, mely – bár megfelelő jogosultságokkal pillanatok alatt elvégezhető - nem tartozik az általános kliens-szerver oldali munkafolyamatok közé.

33 http://www.jsbeautifier.org
34 http://procssor.com
35 http://www.cssbeautify.com

Komponensek tömörítése Gzip segítségével

Az erőforrások tömörítése egyértelműen csökkenti a hálózaton keresztül küldött adat mennyiségét. A legtöbb modern böngésző támogatja a HTML, CSS és JavaScript fájlok tömörítését, így a megfelelő tömörítés használatával jelentős mértékben csökkenthető az egyes webtartalmak letöltési ideje.

A tömörítés lehetőségének biztosításához a webszervert szükséges konfigurálnunk. A `Content-Encoding` fejlécet *gzip* vagy *deflate*[36] módba állítva akár az összes tömöríthető erőforrásra megadhatunk szabályokat. Ismét a *.htaccess* fájlban kell dolgoznunk, a szerverünkön pedig a *mod_deflate* modul aktiválására lesz szükségünk. A tapasztalatunk azt mutatja, hogy az `<IfModule mod_deflate.c>` direktívát mindenképp érdemes használni a szabályok összeállításának elején, máskülönben Internal Server Error hibaüzenetet kapunk inaktív *mod_deflate* modul esetén.

Az alábbi pár sorban leellenőrizzük a modul meglétét, majd deflate tömörítési engedély adunk ki a HTML, CSS és JavaScript állományokra. Természetesen a teljesség kedvéért a sima szöveges, valamint az XML dokumentumokra is beállítjuk a tömörítést.

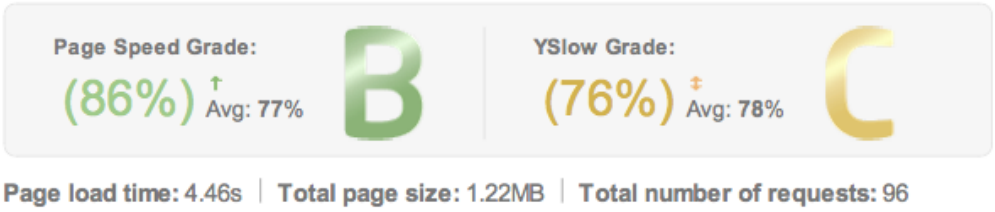
```
<IfModule mod_deflate.c>
# tömörítés: HTML, JavaScript, CSS, valamint text és XML
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
</IfModule>
```

A fenti módszer igen jó hatásfokkal bír, ugyanakkor a betömörítés és kitömörítés folyamata időigényes, így egy bizonyos fájlméret alatt több időt veszítünk a kompresszálási folyamattal, mint amit azzal nyerünk, hogy kisebb méretű adatcsomag utazik a hálózaton. A Google tapasztalatai azt mutatják, hogy 150 és 1000 bájt között határozható meg az a fájlméret, ami alatt már nem érdemes a tömörítést alkalmazni.

36 http://www.ietf.org/rfc/rfc1951.txt

Az optimalizálás eredménye

A fenti optimalizálási feladatokat elvégezve a HTTP kérések számát 35 százalékkal sikerült csökkentenem, míg a weboldal mérete az eredeti oldal 32 százalékára csökkent. A módosítások előtt és után elvégzett teszt eredményének kivonatát a 22. és 23. ábra szemlélteti.



22. ábra: Pontszám optimalizálás előtt

23. ábra: Pontszám optimalizálás után

Az optimalizálási folyamat eredményeként a weboldal nem csak a második legjobban optimizált weboldal lett a magyar piacon, de a világ vezető utazási portáljainak átlaga fölé is emelkedett. Olyan oldalakat is megelőzött, mint a Yahoo vagy a Priceline, az összesített listában a 12. helyről a 8. helyre került (3. táblázat).

#	Összesített – utána	PageSpeed	Yslow	Átlag
1	facebook.com	98	97	97,5
2	google.com	98	94	96
3	tripadvisor.com	94	92	93
4	youtube.com	95	86	90,5
5	amazon.com	94	82	88
6	booking.com	93	82	87,5
7	limba.com	87	83	85
8	*belfoldiszallasok.hu*	*88*	*81*	*84,5*
9	itthon.hu	90	78	84
10	yahoo.com	89	77	83
11	priceline.com	88	78	83
12	xe.com	89	73	81
13	iranymagyarorszag.hu	83	77	80
14	utisugo.hu	86	69	77,5
15	szallasinfo.hu	76	72	74
16	expedia.com	81	60	70,5

3. táblázat: Összesített eredmény az optimalizálás után

Nyilvánvaló, hogy további finomhangolás után még hatékonyabbá tehető a weboldal működése, így a hatékony módszerek, technikák kutatása a jövőben is folytatódik.

Eredmény

Mintegy összegezve az alábbi folyamatot érdemes követnie egy szoftverfejlesztőnek annak érdekében, hogy akár egy meglévő akár egy újonnan készülő webportált gyorssá tegyen.

Előkészületek webszerverre való feltöltés előtt

- képek átméretezése, kicsinyítése
- képek kombinálása CSS Sprite-ok segítségével
- CSS fájlok minimalizálása
- CSS fájlok betöltése a tartalom elején
- JavaScript fájlok kombinálása
- JavaScript fájlok minimalizálása
- JavaScript fájlok betöltése a tartalom végén
- HTML kódban képek dimenzióinak megadása

Webszerver beállítások

- tömörítés engedélyezése
- gyorstárazás engedélyezése
- aldomain létrehozása a statikus tartalmak kiszolgáláshoz

Reszponzív, tartalomfüggő megjelenítés

Gazdag Internet Alkalmazás (Rich Internet Application)

Az elmúlt pár évben az IT technológiák és a számítástudomány fejlődése igencsak a web felé orientálódott, a Webet hamar a jövő fejlesztési platformjaként aposztrofálták. Ugyanakkor az már nem egyértelmű, hogy a „web, mint platform" igazából milyen értelemmel is bír.

Alapvetően a szakirodalom háromféle fogalmi megközelítést tárgyal, az első ezek közül a web-alapú felhasználói felületeket tekinti platformnak, mely arra az alapvetően egyszerű ötletre épül, hogy a meglévő alkalmazásokhoz web-alapú interfészt biztosít a felhasználók számára. Az önálló, vagy platform-specifikus kliens-szerver alkalmazások készítése helyett.

A második nézőpont a böngészőt magát tekinti a fejlesztés platformjának, ahol ezen kliens-szerver platform egyre gazdagodó eszköztára és szolgáltatásai lehetővé teszik új osztályok és alkalmazások használatát, gyakran áthelyezve szerver oldalról kliens oldalra az összes alkalmazás logikát.

A harmadik megközelítés szerint a web-alapú kliens oldali környezetet tekintjük a fejlesztés platformjának; ezt a megközelítési módot nevezik gyakran Rich Internet Application (RIA) – nek [75].

A Rich Internet Application (RIA), mint fogalom definiálására számos kísérletet láthatunk a szakirodalomban, a technológiára való igény és így a fogalom már 2002-ben megszületett a Macromedia háza táján [76], azóta számos szerző és kutató foglalkozott a témával, csak hogy néhányat említsünk:

Bozzon és társai 2006-ban a következőt írták [77] :

„A RIAk a web alapú rendszerek olyan változatai, melyek kifinomult felhasználói felületeket biztosítanak összetett folyamatok és adatok ábrázolására, minimalizálják a kliens-szerver közötti adatforgalmat valamint az interakciós és megjelenítési réteget a

szervertől áthelyezik a kliens oldalra. A RIAt tipikusan - néhány inicializáló adattal együtt - a kliens tölti be; majd menedzseli az adatmegjelenítést és eseményfeldolgozást és kommunikál a szerverrel, amint a felhasználó további információra kíváncsi, vagy amikor adatot kell elküldenie számára."

Majd 2008-ban Santiago Meliá és társai [78], valamint Paul és Harvey Deitel [79] az alábbiakat fogalmazta meg:

"A web alkalmazások felhasználói felületei hagyományosan korlátozott lehetőségekkel bírnak a használhatóság és interaktivitás terén. Ahhoz, hogy ezeken a korlátokon túllendüljünk, újfajta webalkalmazások jelentek meg - a nevük Rich Internet Application (RIA) - ,melyek jóval gazdagabb és jóval hatékonyabb, az asztali alkalmazásokhoz hasonló grafikus komponenseket kínálnak."

A RIAk olyan web alkalmazások, melyek a reszponzivítást, az asztali alkalmazásokhoz hasonló "gazdag" szolgáltatásokat és funkcionalitást nyújtják. A kezdetei Internet alkalmazások csak egyszerű HTML grafikus felhasználói felületeket (GUI) támogattak. Mivel ezek csak alapvető funkciók kiszolgálására voltak alkalmasak, közel sem nyújtották az asztali alkalmazásokhoz hasonló kinézetet és érzetet. A RIAk, a mai, jóval fejlettebb technológiák eredményei, melyek nagyobb reszponzivítást és fejlettebb GUI-k készítését teszik lehetővé."

Végül Busch és Koch 2009-es Technical Reportjukban az alábbiak szerint foglalták össze a RIA lényegét [80]:

„A RIAk olyan web alkalmazások, melyek kliens és szerver oldalon egyaránt feldolgozható adatokat használnak. Mindemellett az adatcsere asszinkron módon történik, a kliens reszponzív marad, miközben folyamatosan újrakalkulálja és frissíti a felhasználói felület bizonyos részeit. "

Mindhárom jellemzésből kitűnik, hogy a "gazdagság", azaz ami megkülönbözteti a korai web alkalmazásokat a mai alkalmazásoktól az asztali alkalmazásokhoz hasonló kifinomult, kényelmes és rengeteg sokoldalú funkcióval bíró felhasználói felületet jelent. A folyamatos reszponzívitást, azaz a felhasználói beavatkozásra történő azonnali reagálás képességét az asszinkron kliens-szerver kommunikáció biztosítja. Ennek motorja és lelke nem más, mint az AJAX, aki biztosítja a lehetőségét annak, hogy minimalizált, csak a legszükségesebb adatok mozogjanak a kommunikációs csatornán, mindez a háttérben, így a GUI képes bármikor reagálni a felhasználó beavatkozására, igényeire.

Az AJAX mozaikszót eddig is többször említettem, a továbbiakban is sokszor előkerül, itt az ideje, hogy ismertessem, miről is van szó.

Az AJAX

Az AJAX - Asynchronous JavaScript And XML - nem egy új technológia, hanem több különböző, önállóan is remekül használható webtechnológia összessége. A mozaikszó elsőként Garett 2005-ös cikkében jelent meg [81], az iménti definíció is az ő tollából származik.

Maga a technológia, mely szerint a szerver oldali alkalmazás képes a klienssel adatot cserélni a weboldal újratöltése nélkül is, már 2002-ben létezett *remote scripting* néven [82], ugyanakkor igazán népszerűvé akkor vált, amikor elkezdték a Google mérnökei is használni; gondoljunk itt akár a Google Suggest, a Gmail vagy a Google Maps szolgáltatásokra. Bár az AJAX számos meglévő technológia összessége, az eredetileg leírt és javasolt technológiai csomagból [83] elegendő csak 3 komponenst használnunk. Szükségünk van

- XMLHttpRequest[37] objektumra a kliens-szerver közötti asszinkron adatkommunikációért - ebben rejlik az AJAX lényege
- A DOM[38] (Document Object Model) -re a dinamikus megjelenítés, struktúraváltoztatás lehetőségének biztosítására
- valamint JavaScriptre a kliens oldali adatfeldolgozásért, az adatok megjelenítéséért, valamint az előző technológiák összefogásáért

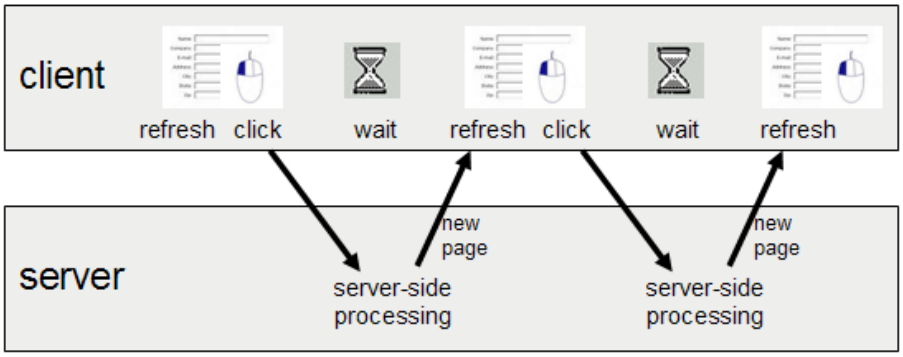
A technológia lényege, hogy az AJAX az XMLHttpRequest JavaScript objektumot használva úgy tud adatot cserélni a webszerverrel, hogy az adott weboldal nem töltődik újra. Az Ajax asszinkron adatcserét hajt végre a böngésző és a webszerver között (HTTP request-ek segítségével), ezáltal elérve azt, hogy a teljes weblap letöltése helyett, annak csak bizonyos részinformációit frissíti.

37 http://www.w3.org/TR/XMLHttpRequest/
38 http://www.w3.org/DOM/

Szinkron kommunikáció

Ahhoz, hogy ennek jelentőségét megértsük, meg kell vizsgálnunk a klasszikus webalkalmazás kliens-szerver kommunikációját. A kommunikáció - egy életszerű példa alapján - a következőképpen néz ki:

1. Kliens oldalon a felhasználó böngészi a weblapját, kitölt egy űrlapot, majd megnyom egy gombot, vagy rákattint egy linkre. Ekkor
2. A böngészője *http request*-ek segítségével információt küld a webszerverhez.
3. A webszerver dolgozik, (adatokat kér le, számol, autentikál, stb.) majd
4. visszaad egy új HTML lapot a kliensnek, azaz a böngészőnek (24. ábra).



24. ábra: A klasszikus HTML kommunikáció – a felhasználó szemszögéből

Forrás: http://www.openajax.org/whitepapers/

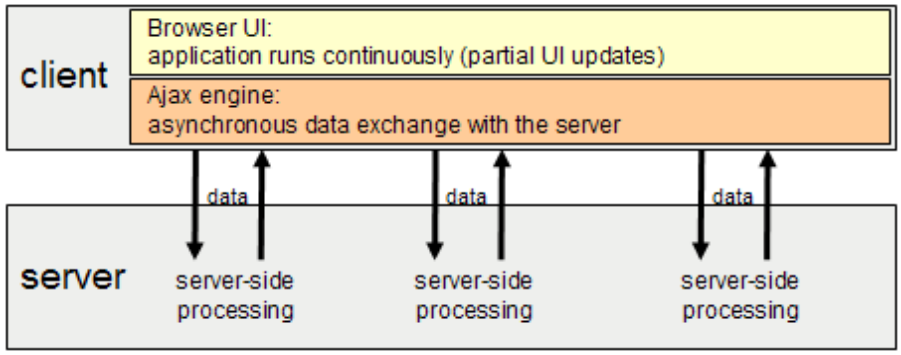
Mi a gond ezzel?

- Az oldal működése megáll, amíg a webszerver felé a kérés le nem zárul. A felhasználó csak ül és várja, hogy megjelenjen a válasz, az új weblap vagy épp az eredmény.
- A teljes oldal letöltődik (képek, szkriptek, stíluslap, HTML elemek) annak ellenére, hogy annak – az előzőhöz képest – csak egy töredék része változott.
- Ezáltal felesleges és szükségtelen adatforgalom generálódik, amely legfőbb következménye, hogy még tovább várakoztatjuk a felhasználót.

Ez pedig a mai világban megengedhetetlen, az idő ugyanis az egyik legfontosabb tényező, mely befolyásolja az online vásárlási döntéseinket.

Asszinkron kommunikáció

Itt lép képbe az AJAX, és a nevében hordozott asszinkron kommunikáció. Az asszinkronitás ebben az esetben arra utal, hogy az aktuális oldal megjelenítése és a háttérben, a szerverrel zajló kommunikáció egymástól független. Bár a kliens-szerver architektúra mindkét esetben ugyanaz, egy nagyon fontos különbség van a két technológia között. A kliens oldalon ott ül az Ajax motor, mely irányítja a szükséges adatcserét a webszerver és a webböngésző között. A kulcsszó ebben az esetben a 'szükséges', mivel alkalmanként a teljes weblapnak mindössze egy kis szelete utazik a kommunikációs csatornán és az is csak akkor, ha szükséges (25. ábra).



25. ábra: Az AJAX kommunikáció

Forrás: http://www.openajax.org/whitepapers/

- Míg az AJAX a háttérben várakozik, hogy az adatok megérkezzenek a szervertől, addig a felhasználó nyugodtan tudja használni az épp aktuális oldalt.
- Amint megérkeztek az adatok, az AJAX motor az új tartalmat megjeleníti a weblapon úgy, hogy nem a teljes oldalt módosítja, hanem annak csak azon részeit, amely frissült, változott.

Azáltal, hogy nem a teljes oldal adatai mozognak a böngésző és a webszerver között, hanem csak a változtatott adatok, jóval gyorsabb a kommunikáció és az új információk megjelenítése.

Ráadásul teszi mindezt az eddig megszokott „frissítéskori villanás” (flickering / blinking) nélkül, így a felhasználó alkalmasint észre sem veszi,

hogy már új adatokat tartalmaz az általa megnyitott weboldal.

Az AJAX motor felépítése

Az általános technológiai ismertetés után nézzük meg, hogyan épül fel egy ilyen AJAX motor.

Az AJAX népszerűségének hála mára már számos előre gyártott kódkönyvtárat találhatunk az Interneten, ám jelen értekezésben mégis egy saját AJAX motort készítettem. Egyrészt a kutatás egyéb részeiben is használom, hivatkozok rá, másrészt ha lehetőség van rá, inkább egy saját fejlesztésen keresztül ismertetem a működését. Ezen gondolatmenet alapján következzék a saját AJAX motor.

Első lépésként létre kell hoznunk egy XMLHttpRequest példányt az alábbi JavaScript utasítással.

```
req = new XMLHttpRequest();
```

Természetesen mint mindig, most is ügyelnünk kell a böngészőkompatibilitásra. Az Internet Explorer másképp nevezi ezt az objektumot, mint a Mozilla alapú rendszerek, így az alábbi függvényünk lesz alkalmas egy XMLHttpRequest példány létrehozására:

```
function UjHttpRequest()
{
   var req = false;
   try {
       req = new ActiveXObject('Msxml2.XMLHTTP');
        }
   catch (e1) {
       try {
             req = new ActiveXObject('Microsoft.XMLHTTP');
           }
        catch (e2) {
           try {
               req = new XMLHttpRequest();
               }
```

```
            catch (e3)
              {
              req = false;
              }
          }
     }
    return req;
}
```

Második lépésként várakoznunk kell a webszerver által küldött válaszra. Erre szolgál az XMLHttpRequest objektum *onreadystatechange* attribútuma, amelyhez rendelt függvény akkor fut le, ha az adatok megérkeztek a klienshez.

```
http_req.onreadystatechange = function()
{}
```

A fenti függvényünk fogja tartalmazni a harmadik lépésként elkészített lekérést. Ehhez a korábban inicializált XMLHttpRequest objektum metódusait és attribútumait fogjuk használni.

Attribútumok
readyState: a lekérési folyamat állapotát jelöli és 4 különböző értéket vehet fel: 0 – még nem indult el a lekérés, 1- a kapcsolat létrejött, 2 – lekérés elküldve, 3 – feldolgozás. 4 – kész

responseText: a szervertől megkapott szöveg, lényegében maga az érdemi adat, amire várunk

Számunkra akkor érdekes a dolog, ha már megjöttek az új adatok, azaz a lekérési folyamat befejeződött, kész, így az alábbi kódra van szükségünk:

```
if (http_req.readyState == 4)
{
document.getElementById("tartalom").innerHTML =
http_req.responseText;
}
```

A fenti kódrészlet a *tartalom* azonosítójú DOM elemben - ami jellemzően egy <p> vagy még inkább egy - megjeleníti a webszerver által visszaadott értéket. A visszaadott értéket a *responseText*-ben szerepel, ami a *http_req* nevű objektum attribútuma, ahol a *http_req* objektum egy XMLHttpRequest példány.

Metódusok

open (mode, url, boolean)
Ez az a metódus, amely kommunikációs csatornát nyit a webszerver felé, méghozzá az alábbi paraméterekkel:

- *mode* – az adatátvitel módja, lehet GET vagy POST épp úgy, mint HTML űrlapok esetében
- *url* – annak a fájlnak az elérési útja a webszerveren, mely feldolgozza majd a kliens oldal kéréseit
- *boolean* – lehet true vagy false, ez a kapcsoló jelzi, hogy szinkron vagy asszinkron módon akarunk a webszerverhez kapcsolódni. Természetesen mi asszinkron módon fogunk csatlakozni, hisz ez az Ajax lényege.

send (string)
A send metódussal tudjuk elküldeni a kérésünket az *open* metódus *url* paraméterében megadott programnak.

A használt metódusok az alábbi kódrészletet alkotják:

```
http_req.open("POST", url, true);
http_req.setRequestHeader("Content-Type","application/x-www-
form-urlencoded; charset=utf-8");
http_req.send(adat);
```

Az *open* metódussal megnyitottuk az asszinkron kommunikációt az *url* paraméterben megadott program felé, majd a szabványos kommunikáció érdekében – a fenti példában látható módon - beállítottuk a HTTP kérés header részét is a *setRequestHeader* nevű metódussal. Ezek után pedig a kliens a *send* metódussal elküldte a webszervernek az *adat* változóban megadott adatokat.

Nézzük tehát egyben, hogyan néz ki az AJAX motorunk.

```
function AjaxMotor(url, adat)
{
 var http_req = UjHttpRequest();
 http_req.onreadystatechange=function()
  {
   if (http_req.readyState == 4)
    {
  document.getElementById("tartalom").innerHTML = 
http_req.responseText;
    }
  };
  http_req.open("POST", url, true);
  http_req.setRequestHeader("Content-Type",  "application/x-www-
form-urlencoded; charset=utf-8");
  http_req.send(adat);
}
function UrlapKuld(kid,vid)
 {
  var adat = "kerdesid="+kid+"&valaszid="+vid;
  AjaxMotor("lista.php", adat);
 }
```

A programkód végén található még egy *UrlapKuld* nevű függvény, mely a webszerver felé küldendő adatcsomag összeállításáért és az *AjaxMotor* függvény meghívásáért felelős.

A fejlesztői környezet építőkövei

Joggal merül fel a kérdés, hogy a mai elvárásoknak megfelelő, gazdag felhasználói élményt nyújtó webrendszerek fejlesztéséhez kapunk e valamilyen segítséget, eszközkészletet, útmutatást?

Bár a látványos és gyönyörű grafikai elemekért a programozási munkától független grafikus szakemberek munkáját dicséri, az interaktív és egyre lenyűgözőbb funkcionalitással bíró, GUI-n megjelenő komponensekért az a számos JavaScript keretrendszer a felelős, amelyek nélkül ma már komplexebb webes projekt nem készül.

Érdemes megjegyezni, hogy az eddigi kutatások, és a RIA megjelenése is kizárólag a böngészőbe épített úgynevezett plugin-ek[39] segítségével látta elképzelhetőnek a gazdag felhasználói élmény megvalósulását. Így Java Applet, JavaFX, Adobe Flash, Flex vagy Microsoft Silverlight eszközrendszerét használták, modellezték, kutatták. Ugyanakkor a mobil platform erőteljes térhódításának következtében, ahol ezen plugin-ek nem, vagy csak korlátozott módon használhatóak, illetve magának a ténynek, hogy a plugin-ek nem feltétlenül vannak telepítve az adott felhasználó gépén, a legújabb elemzések több RIA kategóriát alkottak meg, így létezik szkript-alapú, plugin-alapú és web-alapú asztali alkalmazás.

A plugin alapú web alkalmazás és a web-alapú asztali alkalmazás közötti különbség nem mindig egyértelmű, elég csak egy Silverlight-tal készített alkalmazásra gondolnunk, mely éppúgy futtatható böngészőben, mind önállóan.

Ráadásul, ahogy az előző bekezdésben is jeleztük, a plugin-alapú rendszerek erősen platformfüggőek, bizonyos esetekben egyáltalán nem működnek. Épp ezért a saját üzleti életből vett tapasztalatok, valamint a legújabb trendeknek megfelelően a plugin-mentes, szkript-alapú rendszerekkel foglalkozunk, azokat tárgyaljuk. Ezen rendszerek alapjai pedig egyértelműen a JavaScript (Ajax) keretrendszerek.

Kliens oldali keretrendszerek

A nyílt forráskódú JavaScript keretrendszerek használata egyre népszerűbb trend a webes alkalmazásfejlesztés során, a W3Techs.com - 2014 januárjában a leglátogatottabb 10 millió weboldalon végzett - elemzése alapján kijelenthetjük, hogy a mai weboldalak 61,8%-a használ valamilyen JavaScript keretrendszert, és ezek 92,9%-a a jQuery eszköztárára épít (4. táblázat) [84].

39 http://whatis.techtarget.com/definition/plug-in

Név	Piaci részesedés	Forrás
Jquery	92,9%	http://jquery.com/
MooTools	8,1%	http://mootools.net/
Modernizr	7,1%	http://modernizr.com
Prototype	4,5%	http://prototypejs.org/
ASP.NET Ajax	3,7%	http://www.asp.net/ajax/
Script.aculo.us	3,5%	http://script.aculo.us/
YUI	1,4%	http://developer.yahoo.com/yui/
Spry	0,8%	http://labs.adobe.com/showcase/spry/
Shadowbox	0,8%	http://www.shadowbox-js.com
Dojo	0,2%	http://dojotoolkit.org/
Underscore	0,2%	http://underscorejs.org
Ext JS	0,1%	http://www.extjs.com/
BackboneJS	0,1%	http://backbonejs.org
KnockoutJS	0,1%	http://knockoutjs.com
AngularJS	0,1%	http://angularjs.org

4. táblázat: A legnépszerűbb JavaScript keretrendszerek

*Egy adott webportál többféle keretrendszert is használhat

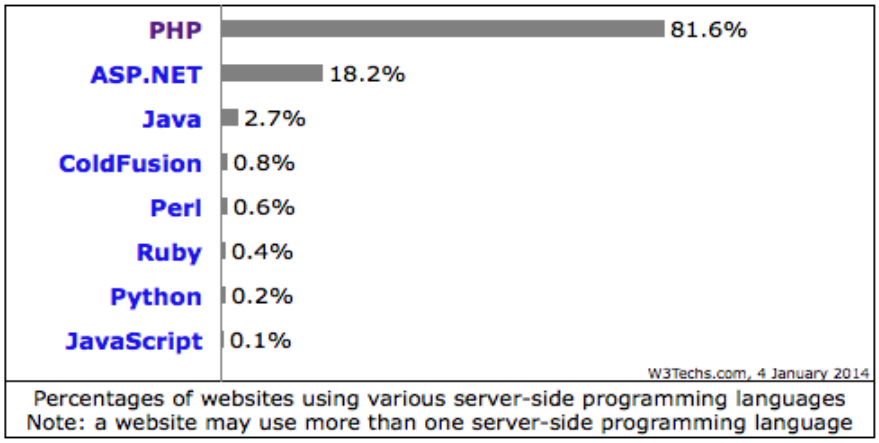
Nem véletlen, hogy olyan nagyvállalatok, mint a Microsoft, az Amazon, a The Guardian vagy a Fox News mellett [85] az elmúlt évek fejlesztői munkái során mi is ezt a keretrendszert választottuk a fejlesztések eszközének. Szigorúan véve a jQuery nem keretrendszer, mint inkább egy integrálható szkriptkönyvár, melyből saját magunk építhetünk fel tetszőleges MVC architektúrát. Amennyiben – érhető módon – mégis MVC keretrendszert szeretnénk használni, igen nagy segítséget nyújt a döntésben a TodoMVC[40], ahol igen széles kínálatból választhatjuk ki a kedvünkre való framework-öt.

Szerver oldali keretrendszerek

Szerver oldalon talán még elterjedtebb a keretrendszerek használata, számos igen népszerű és jól használható rendszerrel találkozhatunk. Nem könnyű a webfejlesztő dolga, ha választania kell, a szerver oldali programozási nyelv szűkítheti a kört; kíváncsiak voltunk, melyek a legnépszerűbb nyelvek.

A W3Trends.com 2014 januári statisztikája [86] alapján kijelenthetjük, hogy messze a PHP nyelv az, amely a legnépszerűbb (26. ábra).

40 http://www.todomvc.com



26. ábra: Szerver oldali programozási nyelvek megoszlása

Az igazsághoz hozzátartozik, hogy sok PHP alapú CMS rendszer létezik (Drupal, Joomla, WordPress, stb.) melyek használata lényegesen kevesebb programozási feladatot ró a webfejlesztőkre, mintha a tiszta nyelvet, vagy fejlesztői keretrendszert használna. Ez torzítja a képet, érdemes lenne ezen rendszerek nélkül megismerni a nyelvek eloszlását.

Szerencsére a CMS rendszerek használatára is kapunk statisztikát [87]: ma a weboldalak 35,2%-a használ valamilyen CMS-t. Feltételezve, hogy ezen rendszerek mindegyike PHP alapú, a valós PHP használat ma 46,4 %, azaz még így is messze a legnépszerűbb szerver oldali programozási nyelv.

Ezek alapján a PHP alapú MVC keretrendszereket gyűjtöttük össze. Minden fejlesztőnek és szakmai portálnak megvan a saját top 10 legjobb keretrendszere, a teljesség igénye nélkül felsoroljuk az általunk legnépszerűbbnek tartott open-source keretrendszereket. (CodeIgniter[41], CakePHP[42], Yii[43], Laravel[44], Symfony[45], Zend Framework[46]).

41 http://ellislab.com/codeigniter
42 http://cakephp.org
43 http://www.yiiframework.com
44 http://laravel.com
45 http://symfony.com
46 http://framework.zend.com

Éles környezetben ideális kombinációnak látszik kliens oldalra jQuery alapú JavaScript MVC keretrendszert választani, míg szerver oldalon a felsorolt 6 PHP rendszer valamelyikét.

Szerencsére az integrált MVC modell nem nyelvfüggő, így bármilyen nyelvet is választunk, alkalmazható rá a korábban bemutatott tervezési minta.

Nos, miután kiépült a fejlesztői környezetet mind kliens, mind szerver oldalon, elkezdhetjük a munkát.

Reszponzív vagy adaptív?

Rögtön a legelején tisztáznunk kell a reszponzív és az adaptív megjelenítés közötti eltérést.

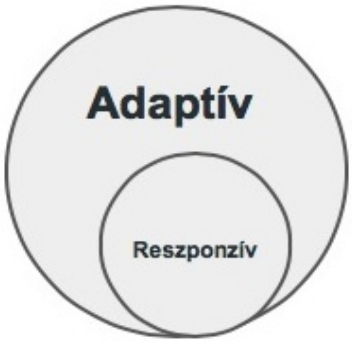
Bár mindkét módszer célja, hogy a webrendszerek különböző mobileszközökön és képernyőméreteken is ugyanolyan felhasználói élményt nyújtsanak, a reszponzív megközelítés szerint a felhasználói felület 'folyékony' (fluid) valós időben alakítható; CSS media lekérdezések segítségével állapítjuk meg kliens oldalon a képernyő aktuális méretét és orientációját, majd ennek alapján alakul át a kinézet. Adaptív megjelenítés esetén a legjellemzőbb képernyőméretekre (360, 480, 720, 960 pixelszélesség) előre elkészítjük a felhasználói felületeket és a szerver ezek közül épp azt aktuálisat, a mobileszköz felbontásához illőt szolgáltatja.

A reszponzív megközelítés ellenzői az adaptív megoldást gyorsabbnak és hatékonyabbnak tartják, hisz adaptív esetben épp az aktuális készüléktípusra, vagy készülékkategóriára készült Nézetet küldjük csak meg a kliensnek.

Az adaptív megoldás ellen szól ugyanakkor, hogy fix képernyőméretekre készül, nincs meg benne az a rugalmasság amit a reszponzív felület kínál.

A Google mégis inkább a reszponzív megközelítést javasolja, ahol a szerver minden eszköznek ugyanazt a HTML kódot küldi ki, majd a kliens oldalon a CSS alakítja a megfelelő méretre a kinézetet [88].

A saját terminológiám ettől eltér, az adaptivitás alatt nem csak a képernyőméret- és eszközfüggő megjelenítést értem, hanem a felhasználó egyéb igényeihez és jellemzőihez kapcsolódó alkalmazkodást. Legyen az nyelvi beállítás, földrajzi lokáció vagy épp nap- vagy évszak. A reszponzivitás az értelmezésem szerint részhalmaza csak az adaptivitásnak és egyértelműen az eszközfüggő felhasználói felület-megjelenítésre utalok vele. A többi paraméter, mely nem a kinézetre, hanem a tartalommegjelenítésre vonatkozik a nagyobb, adaptív halmazba tartozik (27. ábra).



27. ábra: Adaptív struktúra

Nem szabad megfeledkeznünk arról, hogy ma már ugyanúgy megjelenik webrendszerünk okostelefonokon és táblagépeken, mint az autónk konzolképernyőjén, a SmartTV-nk felületén vagy épp a hűtőszekrényünk kijelzőjén.

Implicit adatgyűjtés

Az implicit adatgyűjtés lényege, hogy a felhasználó megzavarása nélkül tudunk információt gyűjteni róla, illetve a készülékéről. Ezt mind kliens, mind szerver oldalon megtehetjük, mindkettőnek megvan az előnye és hátránya. Igyekeztem kiaknázni mindkét technika előnyeit, ezért kombinált módszert alkalmaztam.

Kliens oldali detektálás

Mivel az eszközünk maga a kliens, kézenfekvőnek tűnik, hogy böngészőoldalról kapjuk meg a releváns információkat. Szükséges tehát egyrészt a készülékünk típusát, másrészt annak szolgáltatásait detektálni.

A szolgáltatások detektálására megfelelő eszköz lehet a Modernizr[47] keretrendszer használata, segítségével JavaScript alapokon gyűjthetünk be értékes információkat, míg a készülék azonosítására egyértelműen a HTTP header lesz segítségünkre.

Nem titkolt célom a felhasználói adatok tárolása és későbbi elemzése, ezért a kliens oldalon szerzett információkat is elküldöm a szerver számára, ugyanazzal a módszerrel, amit az Integrált MVC architektúránál már ismertettem.

47 http://www.modernizr.com

Szerver oldali detektálás

Szerver oldalon csak a böngészőtől érkező információkra hagyatkozhatunk, a legpraktikusabb elem ebből a szempontból a `User-Agent` fejléc-elem. Egy tipikus `User-Agent` header Mac OS X 10.9.1 operációs rendszerrel telepített Macintosh számítógépről, Safari 7.0.1 böngészővel megtekintve az alábbiak szerint néz ki:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.73.11
(KHTML, like Gecko) Version/7.0.1 Safari/537.73.11
```

Firefox böngészővel ugyanez:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:26.0) Gecko/20100101
Firefox/26.0
```

iOS 7.0.4 operációs rendszerrel felszerelt - iPhone-ról megtekintve:

```
Mozilla/5.0 (iPhone; CPU iPhone OS 7_0_4 like Mac OS X)
AppleWebKit/537.51.1 (KHTML, like Gecko) Version/7.0 Mobile/11B554a
Safari/9537.53
```

Míg egy Samsung Galaxy SIII-mal, Google Chrome böngészőről:

```
Mozilla/5.0 (Linux; Android 4.3; GT-I9300 Build/JSS15J)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.59 Mobile
Safari/537.36
```

A `User-Agent` (UA) sztring értelmezéséhez saját magunk is írhatunk kódot, vagy igénybe vehetünk üzleti megoldásokat, mint a WURFL[48] vagy a DeviceAtlas[49] projektek, amelyek naprakész adatbázist tartanak fenn a lehetséges UA kombinációkról, készüléktípusokról.

48 http://www.wurfl.com
49 http://www.deviceatlas.com

A saját információgyűjtő modell

A felhasználóról szerzett információkat főbb jellemzőik alapján 3 különböző csoportba osztottam, az egyes csoportokat az alábbiak szerint jellemzem:

Készülék detektálás

Jellemzően a felhasználó hardvereszközével kapcsolatos információk tartoznak ide, így a készülék típusa, kategóriája, böngésző verziója, operációs rendszere, képernyő felbontása, valamint a telepített plugin-ek és betűkészletek.

Környezet detektálás

Környezet alatt nem szigorúan a felhasználó hardveres, mint inkább a valódi életterét kívánjuk feltérképezni. Értékes információ, hogy milyen nyelven beszél, mely országban, városban él, ki az Internet Szolgáltatója milyen sebességű az Internet kapcsolata vagy épp az ő időzónája szerint mi az aktuális, ezredmásodpercre pontos dátum.

Viselkedés detektálás

A viselkedés detektálás a felhasználó látogatói, vásárlási szokásait gyűjti össze. Érdemes megtudunk mikor, milyen időközönként látogatja a webportált, azon milyen oldalakat nézeget, milyen szolgáltatásokat rendel meg, vagy épp nem rendel meg!

Milyen következtetésre juthatunk például abból a tényből, ha

éjszaka van ÉS a készülékének típusa egy gépjármű érintőképernyős felülete.

A legegyszerűbb teendőnk, hogy más, sötétben is jól látható színsémára cseréljük webrendszerünk stíluslapját.

Ettől összetettebb feltételek is megfogalmazhatunk, ha például egy hírportál rendelkezik az általunk megalkotott szenzorokkal:

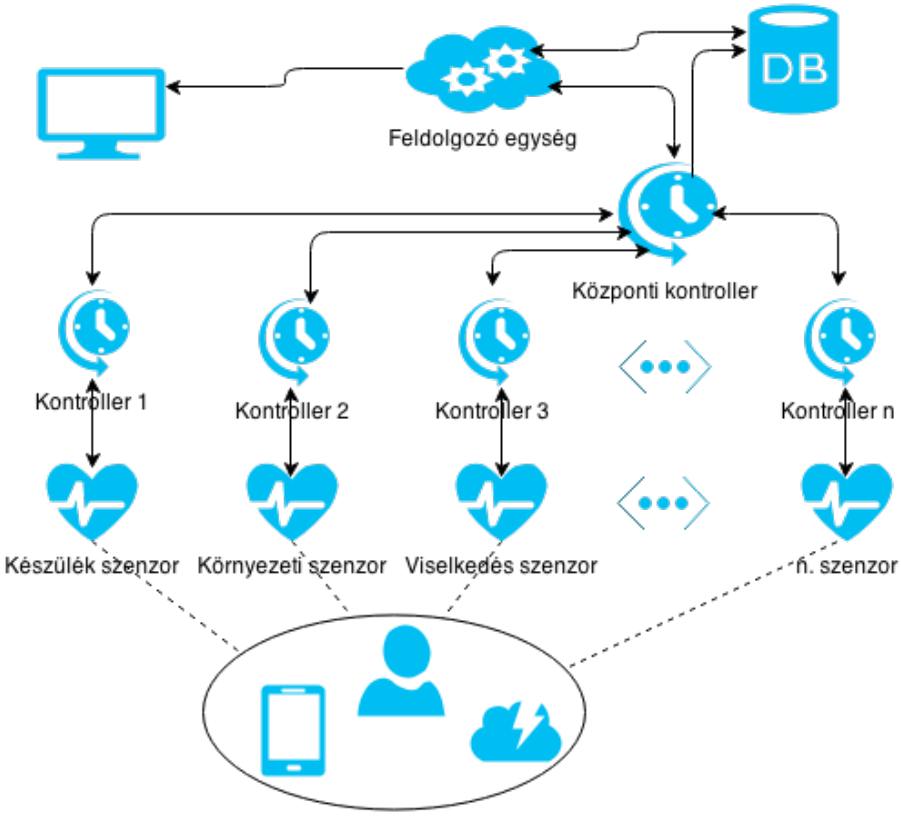
A Debreceni Egyetem hálózatából ÉS portugál nyelvű ÉS orvosi cikkeket olvas ÉS hétköznaponként ÉS 8-18 között

Elég nagy a valószínűsége, hogy egy portugál orvostan hallgató vagy orvos vendégoktatóról van szó.
Ismételten szükséges kiemelnem, hogy ezeket az információkat implicit módon, azaz a felhasználó tudta, engedélye nélkül, észrevétlenül gyűjti a rendszer, ennek minden előnyével és hátrányával.

A három fő kategóriabeli tulajdonságok érzékelésére három szenzort hoztam létre. Mindegyik szenzort felkészítettem néhány tipikus információ detektálására, ezeket az alábbi táblázat foglalja össze.

Szenzor kategória	**Információ kategória**	**Feldolgozó osztály**	**Felhasznált szolgáltatás**
Készülék	Készülékmárka	DeviceBrand	WURFL Cloud
Készülék	Eszköz kategória	DeviceType	WURFL Cloud
Készülék	Képernyőfelbontás	ScreenDetect	Kliens oldal natív
Készülék	Telepített plugin-ek	PluginDetect	PluginDetect.js
Készülék	Operációs rendszer	OSDetect	UserAgent
Készülék	Böngésző típusa	BrowserDetect	UserAgent
Környezet	IP cím	IPDecode	IP-API
Környezet	Földrajzi elhelyezkedés (ország, megye, város, GPS koordináták)	IPDecode	IP-API
Környezet	Internet Szolgáltató	IPDecode	IP-API
Környezet	Nyelvi beállítás	LanguageDetect	UserAgent
Környezet, Viselkedés	Időpont (évszak, hónap, reggel, este)	DateDetect	Szerver oldal natív
Viselkedés	Meglátogatott weboldal	VisitedPageDetect	Szerver oldal natív

5. táblázat: Szenzorok összefoglaló táblázata



28. ábra: Szenzor-architektúra

A szenzorok képességei fejleszthetőek, a moduláris felépítésnek köszönhetően tetszőlegesen bővíthetőek, új információk detektálására taníthatjuk meg őket. A gyakorlati szemléletű megközelítéshez híven 4 tipikus tulajdonságot be is mutatok, az alábbiak szerint:

IP dekódolás – Környezeti szenzor

Az IP cím alapján rengeteg hasznos információt megtudhatunk a felhasználóról, többek között visszafejthetjük földrajzi pozícióját illetve Internet szolgáltatójának nevét. A kliens IP-jét a $_SERVER['REMOTE_ADDR'] szerverváltozó tartalmazza, központi adatbázisok alapján különböző protokollokon keresztül lekérdezhető a hozzátartozó egyéb információ.

Az ingyenes IP-API[50] szolgáltatását vettem igénybe, mely a szintén ingyenes

50 http://ip-api.com

Ip2Location Lite-tól[51] pontosabb és részletesebb információt ad. Az alábbiakban ismertetem az IP cím dekódolására készített osztályt, és a dekódolás eredményét.

```
class IPDecode {
  static $api = "http://ip-api.com/php/";
  static $fields = 65535;
  public $status, $country, $countryCode, $region, $regionName,
$city, $zip, $lat, $lon, $timezone, $isp, $org, $as, $reverse,
$query, $message;
    public static function details($s) {
        $data = self::communicate($s);
        $result = new static;
        foreach ($data as $key => $value)
         { $result->$key = $value; }
        return $result;
    }
    private function communicate($s) {
        if (is_callable('curl_init')) {
            $c = curl_init();
             curl_setopt($c, CURLOPT_URL, self::$api.$s.'?
fields='.self::$fields);
            curl_setopt($c, CURLOPT_HEADER, false);
            curl_setopt($c, CURLOPT_TIMEOUT, 30);
            curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
            $result_array = unserialize(curl_exec($c));
            curl_close($c);
        } else {
    $result_array = unserialize(file_get_contents(self::$api.
$s.'?fields='.self::$fields));
              }
        return $result_array;
```

51 http://lite.ip2location.com

```
        }
}
```

Az IPDecode osztály

Ország:	Hungary (HU)
Megye:	Szabolcs-Szatmar-Bereg
Város:	Nyíregyháza
Koordináták:	47.95, 21.7167
Szolgáltató:	UPC Hungary
IP cím:	188.142.179.**

6. táblázat: WiFi kapcsolat kimenete feldolgozás után

Az IP címek visszafejtésénél ügyelnünk kell arra, hogy nem mindig pontos az érzékelés. Vannak olyan szolgáltatók, akiknél – bárhol is legyünk az országban – a budapesti székhely jelenik meg lokációként. Ennek illusztrálására ugyanarról a mobil készülékről, amely WiFi kapcsolattal az előző táblázatban szereplő eredményt produkálta, 3G kapcsolattal az alábbi információt gyűjtöttük be.

Ország:	Hungary (HU)
Megye:	
Város:	Budapest
Koordináták:	47, 20
Szolgáltató:	Vodafone Hungary Ltd.
IP cím:	130.43.249.**

7. táblázat: 3G kapcsolat kimenete feldolgozás után

Nyelvdetektálás – Környezeti szenzor

Az `Accept-Language` HTTP header-ből könnyűszerrel megállapítható a kliens nyelvi beállítása. Amennyiben ezt meg kívánjuk feleltetni szövegesen is, egy rövid osztályt ismertetek megoldásként.

```
class LanguageDetect {
 static $lang;
 public static function getLanguage($lang) {
  switch ($lang)
   {
    case "en" : $language = "angol"; break;
    case "fr" : $language = "francia"; break;
    case "it" : $language = "olasz"; break;
    case "sk" : $language = "szlovák"; break;
    default   : $language = "magyar";
   }
  return $language;
  }
}
$lang = substr($_SERVER['HTTP_ACCEPT_LANGUAGE'], 0, 2);
$langTxt = LanguageDetect::getLanguage($nyelv);
echo "<hr>Nyelv:". $langTxt;
```

A LanguageDetect osztály

Eszközdetektálás – Készülék szenzor

A detektáláshoz a WURFL Cloud szolgáltatását vettem igénybe, melyhez a regisztráció után egy egyedi API kulcsot szükséges generálnunk. Az ingyenes csomaghoz maximum öt tulajdonság kérhető le, ez persze szabadon variálható. A teljes, közel 500 elemet tartalmazó paraméterlistát a http://www.scientiamobile.com/wurflCapability oldalon érhetjük el.

```
<?
// A WURFL Cloud Kliens osztály
require_once '../WURFLClient/Client.php';
$config = new WurflCloud_Client_Config();
// A WURFL Cloud API kulcs beállítása (saját API szükséges)
$config->api_key = '123456:abcdeuFXjYnIMioqH0mepRhr2L5DwWfJ';

// WURFL Cloud Kliens létrehozása
$client = new WurflCloud_Client_Client($config);

// Eszköz detektálása
$client->detectDevice();?>
```

A WURFL Cloud Kliens használata

Ezek után egyszerű metódushívással

```
echo 'Típus: ' .$client->getDeviceCapability('brand_name');
```

lekérdezhető a telefon típusa, márkája, vagy épp az az információ, hogy mobil eszköz (`is_smartphone`), vagy táblagép (`is_tablet`) e.

A korábban említett Samsung Galaxy S III telefonról megtekintve a fenti kódot tartalmazó oldalt, az alábbi adatokat nyerhetjük ki.

Tulajdonság	Érték	Paraméter
Eszköz típusa:	Samsung GT-I9300 (Galaxy S III)	model_name, brand_name, marketing_name
Eszköz kategória:	mobil	is_smartphone
Operációs rendszer:	Android	device_os
Böngésző verzió:	Android Webkit	mobile_browser
Felbontás szélesség:	720 px	resolution_width
Képernyő szélesség:	360 px	viewport_width
Mutatóeszköz:	touchscreen	pointing_method

*8. táblázat: Samsung Galaxy S III készülék detektált adatlapj*a

Az operációs rendszer és böngészőadatok kinyerésére saját implementációt is használhatunk, a `User-Agent` hordozza ezen információt. A PHP külön metódussal is rendelkezik a lekérdezéshez, ez a `get_browser()`, visszatérési értéke egy asszociatív tömb, benne a megfelelő adatokkal.

Meglátogatott weboldal detektálás – Viselkedés szenzor

A meglátogatott weboldal szerver oldalon a $_SERVER tömb REQUEST_URI kulcshoz tartozó értéke tartalmazza, a látogatás pontos időpontját ugyanezen tömb REQUEST_TIME értéke hordozza. Erre mutat példát az alábbi osztály:

```
class VisitedPageDetect
{
  function __construct($array)
  {
    foreach ($array as $key=>$value)
    {
      $this->$key = $value;
    }
  }
}
$oldal = new VisitedPageDetect($_SERVER);
echo 'A meglátogatott oldal'. $oldal->REQUEST_URI . ' ' .
$oldal->REQUEST_TIME;
```

A VisitedPageDetect osztály

Reszponzív megjelenítés

Miután minden információ a rendelkezésünkre áll, előttünk a lehetőség, hogy a detektált tulajdonságokra reagálva a felhasználó készülékére optimalizált tartalmat szolgáltassunk. Ennek megvalósítására a legújabb módszer a CSS által nyújtott szolgáltatások kiaknázása, a media query-k használata.

Már a HTML4 és a CSS2-es változata is támogatta a médiafüggő stíluslapokat, így például egyazon dokumentum különböző stíluslappal rendelkezhet ha képernyőn, nyomtatásban vagy épp tv készüléken jelenik meg.

A szintaktika a következő:

```
<link rel="stylesheet" type="text/css" media="screen"
href="normal.css">
<link rel="stylesheet" type="text/css" media="print"
href="nyomtat.css">
```

A jelenlegi lehetséges paraméterek: `'all' ,'braille', 'embossed', 'handheld', 'print','projection', 'screen', 'speech', 'tty', 'tv'`, míg az új HTML verziók új médiatípusokat is bevezethetnek, mint például a `'3d-glasses'`.

A media query-k ezt a szintaktikát egészítik ki további kifejezésekkel, melyek alkalmasak arra, hogy egy adott médiatulajdonság létezéséről igaz vagy hamis értéket adjanak vissza.

```
<link rel="stylesheet" media="screen and (min-width:600px))"
href="normal.css" />
```

ugyanez egy CSS stíluslapon belül:

```
@media screen and (min-width:600px) { … }
```

A két szabály ugyanazt jelenti: ha a megjelenítés képernyőre (screen) történik ÉS a szélessége legalább 600 pixel, akkor alkalmazni kell a normal.css stíluslapban, illletve – a második példa alkalmazása esetén – a {} között lévő stílusdefiníciókat.

A szabvány részletesen ismerteti a lehetséges média szolgáltatásokat [89], igen jól támogatja a mobil eszközökre történő optimalizálást; még azt is képes figyelni, hogy a mobil készüléket állítva (portrait) vagy fektetve (landscape) használjuk.

```
@media all and (orientation:landscape) and (min-width: 400px)
and (max-width: 700px) { … }
```

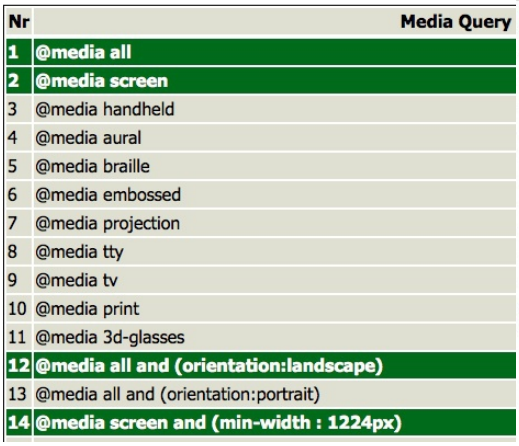
Jelentése: A stílusdefiníciókat akkor kell végrehajtani, ha a készüléket fektetve használják és a képernyő szélessége 400 és 700 pixel között van.

A CSS Media Queries[52] ötletétől inspirálva elkészítettem egy saját CSS média lekérdező modult[53] 40 különböző paraméterre. Az első tesztet egy asztali számítógépről végeztem (29. ábra), míg a másikat egy iPhone készülékről (30.

52 http://cssmediaqueries.com/overview.html
53 http://www.nagyzsolt.hu/ajax/media_query.php

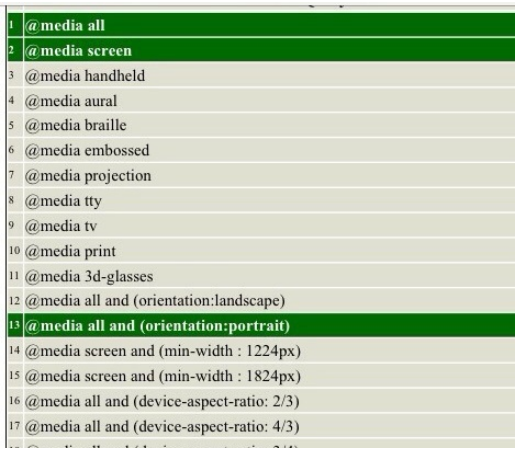
ábra), méghozzá állított (`portrait`) képernyővel.

Az alábbi ábrák szemléltetik a teszt eredményét; a zöld színnel jelzett sorok jelentik azt, hogy az adott sorban megfogalmazott szabály igaz az adott készülékre.



Nr	Media Query
1	@media all
2	@media screen
3	@media handheld
4	@media aural
5	@media braille
6	@media embossed
7	@media projection
8	@media tty
9	@media tv
10	@media print
11	@media 3d-glasses
12	@media all and (orientation:landscape)
13	@media all and (orientation:portrait)
14	@media screen and (min-width : 1224px)

29. ábra: Asztali számítógép CSS media lekérdezés



1	@media all
2	@media screen
3	@media handheld
4	@media aural
5	@media braille
6	@media embossed
7	@media projection
8	@media tty
9	@media tv
10	@media print
11	@media 3d-glasses
12	@media all and (orientation:landscape)
13	@media all and (orientation:portrait)
14	@media screen and (min-width : 1224px)
15	@media screen and (min-width : 1824px)
16	@media all and (device-aspect-ratio: 2/3)
17	@media all and (device-aspect-ratio: 4/3)

30. ábra: iPhone készülék CSS média lekérdezés

Sajnos a képernyőmentés nem képes a teljes listát visszaadni, mindenesetre látható, hogy mind az asztali számítógép, mind az okostelefon képes kezelni az az `all` és `screen` tulajdonságokat. A számítógép fekvő monitorral (`orientation:landscape`) rendelkezik, míg a mobilkészülék álló kijelzővel (`orientation:portrait`) tekintette meg az adott oldalt. Továbbá az asztali gép képes akár 1224 pixel széles felbontásra is, ez a tulajdonság a telefont nem jellemzi.

A fenti tulajdonságlista tetszőlegesen bővíthető, éles környezetben nyilvánvalóan nem írja ki a rendszer a felhasználó számára a lekérdezések eredményét, mindössze felhasználja azokat a felület reszponzív kialakításához.

Éppúgy, mint az implicit adatgyűjtéshez. Hisz az így begyűjtött információ az AJAX technológia révén akár másodpercenként vándorolhat a klienstől a szerver felé, ahol letároljuk és készséggel felhasználjuk az ajánlói rendszerünk bemeneti adataként.

Új, eddig nem alkalmazott adatgyűjtési technika rendszerünkben a CSS média lekérdezések felhasználása.

CSS media lekérdezés eredményének feldolgozása

Az alapötlet az, hogy a médiatulajdonságokat tartalmazó táblázatot végigpásztázzuk, és megragadjuk azokat a sorokat, melyeket a detektálás során átszíneztünk. A táblázatunk halványbarna hátterű sorai

CSS

```
tr {
background-color:#E0E0D1;
}
```

HTML

```
<tr id="r1"><td>1</td><td>@media all</td></tr>
```

ugyanis zölddé változnak, ha megfelelnek az egyes feltételeknek. Így például a

CSS

```
@media all{ #r1{ background-color: #006B00; color: #FFFFFF;
font-weight:bold } }
```

media lekérdezés valós időben változtatja meg az első, `@media all` szöveget tartalmazó sor betűszínét fehérre .

A szín lekérdezésére írt JavaScript kódrészlet

```
if ( document.getElementById("r1").style.color == "rgb(255,
255, 255)" )
{
// teljesült a feltétel, küldhetjük az információt a szervernek
} else
{
// nem teljesült, de ez is hasznos információ lehet
}
```

A detektálás eredményeképp immár kézzel fogható adat áll a rendelkezésünkre, ezt a korábban már ismertetett JSON formátumban egyszerűen megküldhetjük az alkalmazásszerverünk részére.

Adatvédelem szakértői szemmel

A számítógépek rengeteg adatot tárolnak a felhasználóról úgy kliens, mint szerver oldalon. Egyetlen kattintás, egyetlen másodperc egy weboldalon és a webportál tulajdonosa több száz különböző tényt rögzített a látogatóról. Ehhez még az sem kell, hogy intelligens szenzorokkal ruházza fel az oldalt, elég csak olyan webforgalom analizáló rendszereket használnia, mint a Google Analytics.

Bármelyik módszerről is legyen szó, illúzióink ne legyenek; a mindennapos adatgyűjtés hozzá tartozik az Internetes életünkhöz.

Ez rendjén is van, míg az adataink illetéktelen kezekbe nem kerülnek. Bár láttuk az előző fejezetben, hogy milyen módszerekkel, milyen típusú adatokat képesek rólunk összegyűjteni a webportálok, a szervereken tárolt adatok miatt kevésbé kell aggódnunk. Egyrészt fizikailag is jól védett helyen vannak, másrészt a webszolgáltatókra szigorú haza és nemzetközi szabályozások vonatkoznak az adatkezelést illetően.

Kliens oldalon viszont a mi felelősségünk, hogy fizikailag óvjuk számítógépünket, illetve a rajta tárolt érzékeny adatokat. A legnagyobb probléma épp ott jelentkezik, hogy a legtöbb felhasználó nincs is tisztában azzal, milyen kockázatoknak van kitéve; igazából azt sem tudja, milyen adatokat tárol róla saját számítógépe.

Az alábbi kutatási eredményt egy saját szakértői kutató munka inspirálta, melyet aztán kiegészítettem a szakma egyéb tapasztalataival. Az elkövetkező fejezetben azt kívánom bemutatni, hogy a kliens oldalon tárolt adatokat miképp lehet kinyerni és azokból milyen információk állapíthatóak meg.

Elméleti háttér

Manapság számos böngésző közül választhatunk, melyek különböznek kinézetükben, szolgáltatásaikban vagy akár sebességükben. Szakértői szempontból viszont egy tulajdonságukban megegyeznek: általános technológiai jellemzőjük, hogy mielőtt megjelenítenének egy weboldalt, azt

először letöltik a webszerverről majd utána jelenítik meg a helyi számítógép böngészőjében. Annak érdekében, hogy a következő alkalommal ugyanezt a weboldalt gyorsabban legyenek képesek megjeleníteni, a webböngészők ezeket a letöltött adatokat megtartják, így az elérhető marad a számítógépen azután is ha a felhasználó már bezárta a böngészőt vagy kikapcsolta a számítógépet. Ezen letöltött webfájlokat *cache*-nek, *history*-nak vagy *temporary internet file*-oknak nevezzük; attól függően, hogy milyen operációs rendszert és böngészőt használunk, ezek a fájlok a számítógépünk különböző helyein tárolódnak.

A Windows Vistától kezdődően, a Windows 7 és 8 operációs rendszereken az Internet Explorer böngésző által tárolt temporary internet file-ok az alábbi mappában lelhetőek fel:

```
C:\Users\<windowsUsername>\Local\Microsoft\Windows\Temporary
Internet Files\Content.IE5\
```

ahol a <windowsUsername> az aktuális felhasználó Windows bejelentkezési neve. A böngészési előzményeket, azaz a korábban meglátogatott weblapok URL-jeit is megtalálhatjuk a közelben:

```
C:\Users\<windowsUsername>\Local\Microsoft\Windows\History\Hist
ory.IE5\
```

Létezik egy *index.dat* fájl ebben a mappában, mely igen hasznos, ha a felhasználó egyébként már törölte a böngészési előzményeit; a fájl feldolgozásával a törölt előzmények kinyerhetőek.

A Mozilla Firefox a böngészési előzményeket egy SQLite formátumú adatbázis táblában tárolja, melyet a

```
C:\Users\<windowsUsername>\AppData\Roaming\Mozilla\Firefox\Prof
iles\<profilMappa>
```

elérési útvonalon érhetünk el.

A Firefox az első indulásakor automatikusan létrehozza a profil mappát, melyben helyet kapnak a böngészési előzményeket (*places.sqlite*), a letöltött fájlok listáját (*download.sqlite)* és a böngésző által tárolt jelszavakat (*key3.db* és

signons.sqlite) tartalmazó állományok. Bár ezek nem sima szövegfájlok, könnyedén belenézhet bárki egy olyan ingyenes szoftverrel, mint az SQLite Database Browser.

A Google Chrome az előzményeket, a letöltött fájlokat és a weboldalakon megadott felhasználói neveinket és jelszavainkat szintén tárolja:

```
C:\Users\<windowsUsername>\AppData\Local\Google\Chrome\User
Data\Default\Web Data
```

Bár az itt található fájlok nem rendelkeznek .sqlite kiterjesztéssel, a fájlok header részét megvizsgálva megállapítottuk, hogy ezek valójában mégis SQLite formátumú állományok.

A nyomok felkutatása

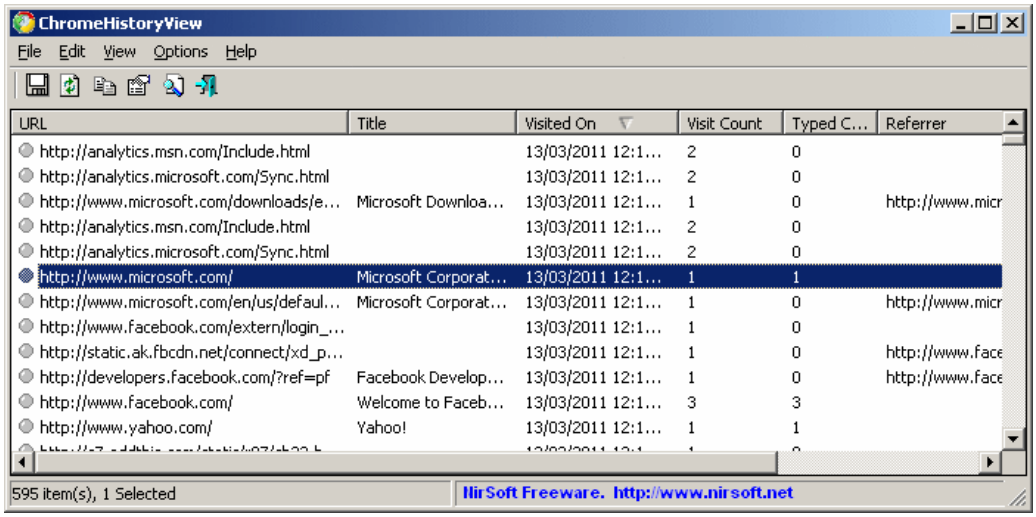
Részletezhetnénk meg tovább, hisz a lista még nem teljes, ám terjedelmi korlátok miatt mi most csak ezekre szorítkoztunk. Szívesebben tárgyaljuk azokat az elérhető szoftvereszközöket, melyek lehetőséget adnak arra, hogy a fenn említett adatokat strukturált formában, automatizált módon nyerjük ki.

Böngészési előzmények feltérképezése

Nincs szükségünk arra, hogy manuálisan bogarásszuk egy-egy számítógép ideiglenesen tárolt fájljait, számos remek alkalmazás létezik a webböngésző-beli aktivitásunk rekonstruálására.

Az Internet Explorerben végzett tevékenységünket az ingyenes IECacheView[54] vagy az üzleti célú Internet Evidence Finder[55] segítségével tudjuk visszaállítani. Természetesen ugyanerre a célra létezik ChromeCacheView[56] és ChromeHistoryView[57] a Google Chrome böngészőhöz, illetve MozillaCacheView[58] és MozillaHistoryView[59] a Firefox böngészőkből való adatok kinyerésére.

54 http://www.nirsoft.net/utils/ie_cache_viewer.html
55 http://www.magnetforensics.com/software/internet-evidence-finder/
56 http://www.nirsoft.net/utils/chrome_cache_view.html
57 http://www.nirsoft.net/utils/chrome_history_view.html
58 http://www.nirsoft.net/utils/mozilla_cache_viewer.html
59 http://www.nirsoft.net/utils/mozilla_history_view.html



31. ábra: A ChromeHistoryView képernyőképe

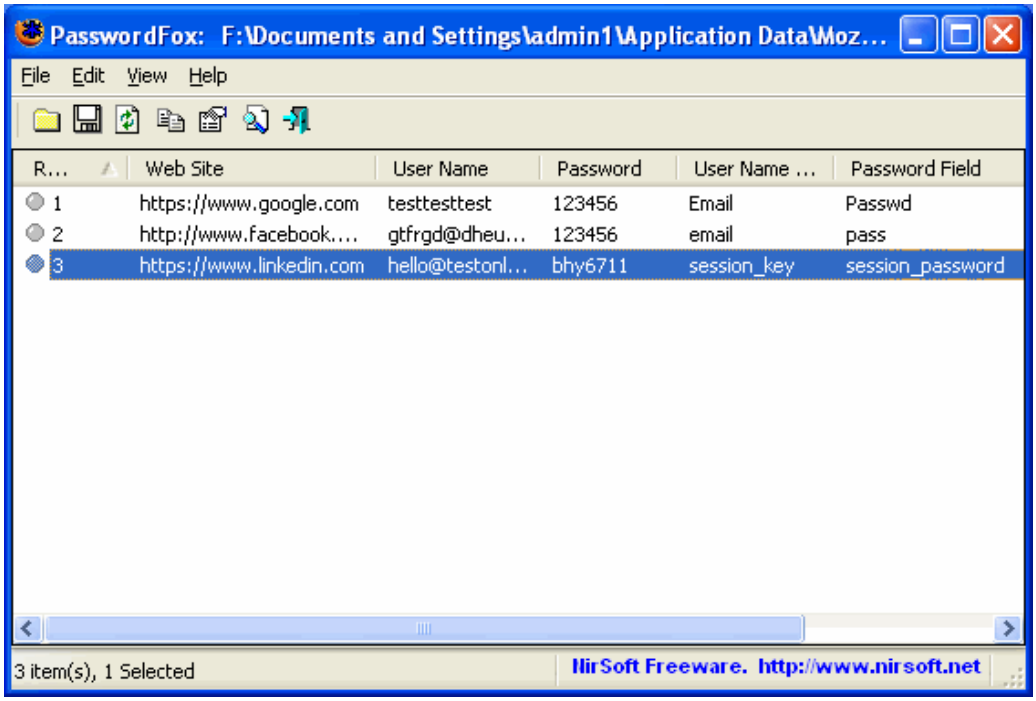
Forrás: http://www.nirsoft.com

A szoftver használatával különböző formátumokba (pdf, xls, html) exportálható listát kapunk arról, hogy mikor, milyen oldalt és hányszor látogattunk meg, sőt azt is megtudhatjuk, mely volt az az előző weblap (referrer) amelyikről az aktuális oldalra vándoroltunk.

Jelszavak kinyerése

Az ettől is érdekesebb és kritikusabb információ, a tárolt felhasználói nevek és jelszavak feltérképezésére szintén léteznek ingyenes megoldások. A IEPassView[60] a PasswordFox[61] és a ChromePass[62] szoftvernevek önmagukért beszélnek, az általuk szolgáltatott információ pedig igencsak impozáns. Megtudhatjuk, hogy mely weboldalra milyen felhasználói névvel és jelszóval léptek be az adott számítógépről (32. ábra).

60 http://www.nirsoft.net/utils/internet_explorer_password.html
61 http://www.nirsoft.net/utils/passwordfox.html
62 http://www.nirsoft.net/utils/chromepass.html



32. ábra: A PasswordFox képernyőképe

Forrás: http://www.nirsoft.com

Az előbb felsorolt ingyenes alkalmazások legnagyobb előnye, hogy semmiféle telepítési procedúrát nem igényelnek, egy egyszerű pendrive-ról futtatható bármelyikőjük. Ha jobban belegondolunk ezek igen veszélyes fegyverek egy kiberbűnöző kezében. Elég csak 2 percre őrizetlenül hagynunk számítógépünket és máris minden bizalmas adatunkat megszerezték.

Védelmi intézkedések

Ahogy az előző fejezetben láthattuk, számos olyan hely létezik számítógépünkön, ahol internetes aktivitásunk lábnyomai feltűnnek; számos olyan eszköz létezik, mellyel könnyűszerrel kinyerhetőek ezek az adatok. A kutatási eredmények három egyszerű módszer megfogalmazására sarkallnak annak érdekében, hogy megnehezítsük az érzékeny adataink kiszivárgását.

Jelszóvédelem

Az első és legfontosabb lépés a jelszavainkhoz kapcsolódik. Tudjuk jól, hogy nem szerencsés egy monitor sarkára ragasztott papírlapkára írva tárolni jelszavainkat, ám láttuk, akkor sem vagyunk biztonságban, ha a számítógépünk maga tárolja ezeket az információkat. Ezért a legfontosabb tanács azon túl, hogy idegeneket ne engedjünk a számítógépünk közelébe; ne kérjük és ne engedjük meg a böngészőnknek, hogy tárolja a bejelentkezési információkat.

Másrészről ha elég óvatosak vagyunk és minden weboldalon más és más jelszóval azonosítjuk magunkat, egy idő után képtelenek leszünk megjegyezni azokat. Szerencsésre mindegyik operációs rendszer kínál hatékony szoftvermegoldásokat arra, hogy biztonságosan tároljuk adatainkat; elegendő egyetlen mesterjelszót megjegyeznünk, mely aztán megnyitja és láthatóvá teszi a mögötte tárolt érzékeny információt.

Hiábavaló a világ legbiztonságosabb széfjében tárolnunk jelszavainkat, ha azok a jelszavakat könnyen ki lehet találni. Egy teljes generáció nőtt fel azóta, hogy az Internet aktív része lett életünknek, mégis, mind a mai napig találkozunk olyan jelszavakkal, mely a tulajdonos háziállatának, családtagjának nevét vagy születési időpontját hordozza.

Ma már szerencsére a legtöbb webrendszer megköveteli a megfelelő biztonsági szintű jelszó megadását, így kis és nagybetű, szám, minimum 8 karakterhosszúság.

Átgondolt közösségi megosztások

Második tanácsként fogadjuk el a tényt: bármi, amit megosztottunk egy közösségi oldalon, feltöltöttünk egy weboldalra vagy átküldtünk Skype-on, kikerül az irányításunk alól. Attól a pillanattól kezdve már nem a saját adatunk többé, bárhol, bármikor feltűnhet, megjelenhet ezután.

Természetesen bízhatunk abban, hogy a barátunknak küldött kép nem kerül ki a kezei közül, vagy abban, hogy a Facebookon csak az ismerőseink láthatják képeinket, ne csapjuk be magunkat; miután bármit is megosztottunk valakivel, onnantól az potenciálisan bárki számára elérhető.

Meglehet, hogy a barátunk nem él vissza a bizalmas információnkkal, ám ha ő maga nem megfelelően védi adatait, a sajátjával együtt a mi érzékeny adatunk is éppúgy veszélybe került.

Épp ezért kiemelt figyelmet szenteljünk arra, hogy közösségi életünk során csak olyan adatot osszunk meg bárkivel, amely később nem jelent problémát, ha nyilvánosságra kerül.

Érdekes tény, és fontos észben tartanunk, hogy a Facebook profilunk egy jövőbeni állásinterjún akár döntő érv is lehet az alkalmazásunk mellett vagy épp ellen. A vezetők és HR szakemberek közel fele nyilatkozott úgy, hogy egy leendő alkalmazott online élete vagy Facebook profilja befolyásolja döntésüket, míg közel 30 százalékuk gondolja azt, hogy a cégnek joga van kirúgni azt a munkatársát, aki nem megfelelő stílusban kommunikál Facebook oldalán [91].

Ezek után igazán érdemes kétszer is meggondolnunk, mit írunk, mihez szólunk hozzá vagy mit osztunk meg a közösségi oldalakon, hisz ez akár a jelenlegi vagy épp a leendő munkánkat, a teljes jövőbeni egzisztenciánkat befolyásolhatja.

A Tiszta lap

A harmadik, egyben utolsó konklúzió: mindig tartsuk észben, hogy a Weben tett minden egyes lépésünk számos lábnyomot hagy maga után számítógépünkön. Épp ezért ne feledkezzünk meg rendszeresen törölni a böngészési előzményeket, az ideiglenesen tárolt internet fájlokat, a Skype vagy egyéb Messenger programok beszélgetéseit. A böngésző- és kommunikációs programok erre lehetőséget adnak, a böngészési előzmények törlése mindössze két kattintás és öt másodperc; szánjuk rá az időt, valóban megéri.

Összefoglalás

Disszertációmban összefoglaltam az elmúlt 5 év kutatómunkáját, mely során igyekeztem minél részletesebben elemezni az intelligens webrendszerekkel kapcsolatban eddig elért nemzetközi eredményeket, felkutatni, felismerni azon területeket, ahol mind tudományos, mind szakmai értelemben érdekes kihívások jelentkeznek, hasznos eredmények születhetnek.

Kutatómunkám során átdolgoztam a webrendszerek fejlesztési folyamatát, majd egy, a mai kor igényeinek megfelelő, MVC alapokon nyugvó tervezési mintát kínáltam, mely jól alkalmazható kliens-szerver oldali keretrendszerek integrálására is. Az intelligens webrendszerek kimeneti adatainak megfelelő megjelenítése kulcsfontosságú a felhasználói élmény és – nem utolsósorban – a piaci profit szempontjából, így egyrészt egy új modellt javasoltam meglévő és új webportálok gyorsabbá tételére, másrészt kidolgoztam egy új, AJAX alapú implicit adatgyűjtő módszert, mellyel igen hatékonnyá tehető az információgyűjtés. A felhasználói interfész reszponzivitása, intelligenciája szintén kiemelt jelentőséggel bír, így javasoltam egy univerzális, reszponzív, tartalomfüggő rendszermodellt, mely tetszőleges témájú webrendszerhez használható, végezetül felkutattam, rendszereztem és bemutattam annak veszélyét, hogy a webrendszerek hol, milyen adatokat gyűjtenek, tárolnak rólunk, felhasználókról.

Értekezésem nem titkolt célja, hogy közelebb hozza egymáshoz az akadémiai kutatásokat az iparban, üzleti életben alkalmazott technológiákkal. Sokszor éri vád a tudományos élet szereplőit, hogy kutatási eredményeik zöme pusztán elméleti jelentőséggel bír, távol áll a valóságtól, nehezen vagy csak évek múlva hasznosítható az iparban. Különösen érzékeny terület ebből a szempontból az informatika, ahol rendkívül gyorsan változik, fejlődik mind a hardver, mind a szoftvertechnológia.

Az elmúlt évek során abban a szerencsés helyzetben voltam, hogy főiskolai oktatói és aktív szoftver- és webfejlesztői munkáim éppúgy hozzájárultak egy-egy témakör részletesebb elemzéséhez, megvilágításához, mint az igazságügyi szakértői vagy a CCNA instruktori feladataim során szerzett tapasztalatok.

Igyekeztem az üzleti életből vett legfrissebb, aktuális problémákat górcső alá venni és azokra olyan megoldást kínálni, mely felhasználja és alkalmazza a tudományos eredményeket; ugyanakkor az elméleti eredmények mellett gyakorlati, azonnal használható és bevethető módszereket is ajánl.

Összességében tehát a kérdés, amire a választ kerestem: Hogyan tehetjük jobbá a mai intelligens webrendszereket?

Summary

In this dissertation I summarized my research work of the past five years, when I focused on the detailed analyses of the international results in intelligent web systems, in order to locate and identify areas, where interesting challenges and results may appear both in scientific and professional way.

In my research, I revised the development process of web systems, offered an MVC based software development design pattern that fits in the nowadays modern methodologies and also suitable for client-server framework integration. Proper display of output data of intelligent web systems is a key regarding the aspect of user experience and market profit, so at first, I proposed a new model for existing and new web portals to make them faster and on the other hand, I developed a new AJAX-based implicit data collection method that improves information collection process more effectively. The intelligence and responsive behavior of a user interface has also considerable importance, thus I have created a universal, responsive, content-aware system, which is suitable for any themed web systems and finally I searched, systematized and presented the risk of web systems; where and how data is collected, stored about us, about the users.

As an unconcealed goal, my dissertation aims to bring closer to each other the academic research area and the technologies used in industry and business life. Academic researchers are often accused of that most of their research results are purely theoretical, away from reality or just takes years to be utilized for the industry. From this point of view information technology, which changes rapidly, is particularly sensitive area, as well as hardware and software technology.

I tried to address the latest, current problems of real industrial and business life, find solutions that use and apply the scientific results; beside the theoretical results, I offer practical, immediately usable and deployable methods as well.

To sum it up, the question that needs to be answered is: How can we make nowadays used intelligent web systems better?

New results

The dissertation consists of six chapters which are closely related to each other where I overview the whole process of a web-based system. In the short reviews of the chapters I describe the initial problem and give my new research results and methods for the discussed problem.

It is just happened that I got insight into the scientific world as a young researcher, therefore it is my pleasure, that foreign researchers cite and use my results and my articles, so please allow me to mention the citations where it is relevant.

In the *first chapter* I reviewed the development process of web-systems, detected the shortcomings of currently used models and developed a new method that meets the demands of today's need.

The *second chapter* highlights the system design from the entire development process, describes in detail the traditional and trendy design patterns used today and then I worked out a new, MVC-based design pattern that is also suitable for the client-server framework integration as well.

The *third chapter* describes the performance shortcomings of current market leader web systems, reviews existing speed increasing technologies and offers a new process model that can be used to build fast web systems or improve existing web portal speed performance.

Chapter four presents an interesting, AJAX-based implicit data acquisition technology, which can make data collection very effective, serves the needs of responsive presentation and offers ideal data collection possibility for the recommender systems.

The *fifth chapter* is intended to be used in making better the component of the software system directly contact with the user. The user interface developed by the improvements of the new model adapts intelligently for its display device, for its type and its screen resolution.

Finally, in the last, *sixth chapter* I explore, analyse and present the privacy risks of existing intelligent systems, I point out what risks are faced in our personal data in the Web 2.0 era.

Adaptive Development Method

Problem

Software development process is complicated and complex. Certain parts of it and the relations among them require modeling in order to make the process modular and easily understandable. The most known and widespread model is the waterfall model. This model is excellent in cases when we know all the system requirements at the very beginning of the development process.

Unfortunately, in real life, customers cannot define exactly what they want at the beginning of the project; requirements change and refine during development process. That is why the waterfall model is declined in many cases during business web development. The waterfall model is not prepared for this kind of changes; when a development phase has been completed, it is almost impossible to make any changes on it.

Solution

We need a method, which is quite flexible and makes it possible to modify the design and system development process and architecture based on continuous customer consultations. Such a method is agile software development, although it cannot be used in many cases. Based on my research-development work, I combined both systems and created a new development model fitting today's modern web development process model, this is the adaptive development method.

I expounded my new results in the following article: *Adaptive Design Process for Responsive Web Development (DOI: 10.13140/2.1.3354.5601)*.

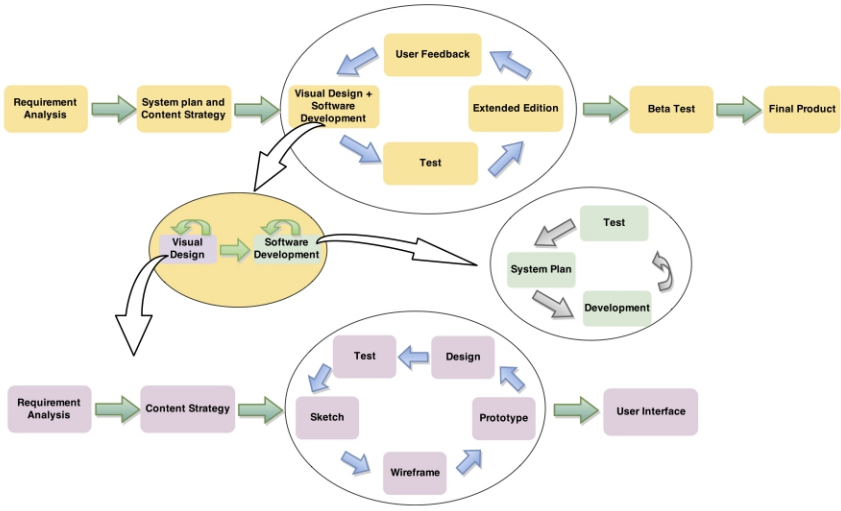


Fig. 1: Adaptive Development Method

Integrated Design Pattern

Problem

Nowadays it is impossible to develop web applications without proper design patterns, as developers must serve both rich client-side programming tasks, as well as usual server side engineering and coding. Whether the client side or server-side programming we are talking about, the volume of the work requires the usage of design patterns. It is cumulatively true for a complex application, where client and server side development is in necessarily indivisible relationship. The most popular design pattern is MVC. In this phase of my research I am searching for the answer whether this pattern is still suitable for nowadays development environments, whether it is possible to offer to the developers a new, better architecture integrating client and server side systems.

Solution

After examination of different design patterns I concluded that the original MVC architecture, or more precisely, its modified, Cocoa version is the perfect initial state for developing a new integrated design pattern. Furthermore, my choice is verified by the fact that MVC frameworks are the most popular

development environments, both on the client and the server side.

Thus the real question is, how we can connect a JavaScript MVC with a PHP MVC framework in such a way that the new, integrated system also fits into the Model-View-Controller architecture.

Approaching from the server side, it is obvious that a View component needs further segmentation, as View complexity makes the development work harder. If we change the View component to a whole client-side MVC, our new system be-comes an M(MVC)C.

In the case of client-side MVC, the Model component of an MVC pattern is a simple HTML code, and the View is a CSS file, (one CSS file means one View), whereas the Controller is the browser itself that assigns HTML with CSS. From another approach, View is a user interface born from a combination of an HTML, CSS and data. JavaScript classes and methods play the role of a Controller, and the Model is the data coming from the web server.

Whichever aspect we take, the Model is the interface where a server-side MVC system could be attached. Based on this result the following figure describes the new design pattern.

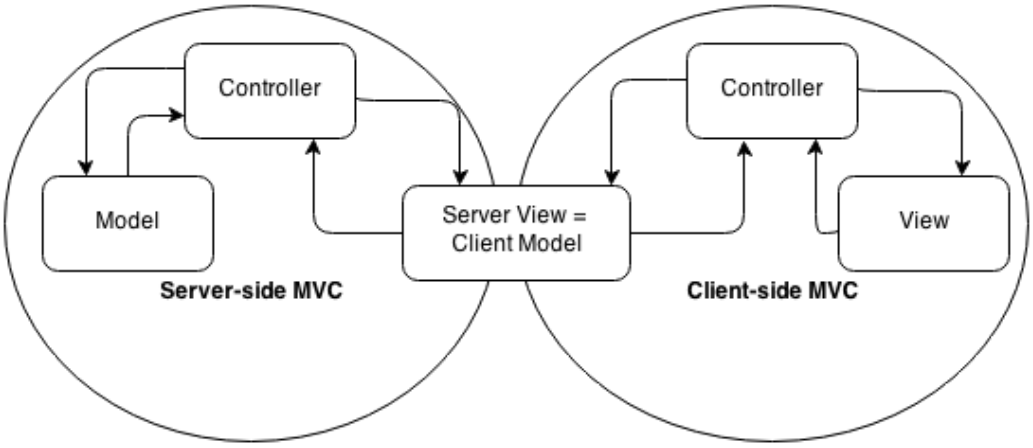


Fig. 2: Integrated Design Pattern

The developed new pattern was presented on SOFA2014 International Conference and my 14 pages article *Integrated Design Pattern for Intelligent Web Applications* was also accepted to appear.

Model for faster web system performance

Problem

As it is mentioned in the introduction, several research confirm: a web system, based on intelligent, excellent mathematical model, worths nothing if the user can hardly wait for displaying the content. Clear proof of the importance of speed is that from 2010 Google has also begun to use webpage speed as an evaluation parameter in its PageRank algorithm. Several interesting book and scientific papers were published about making web pages faster, even the two biggest search engines, Google and Yahoo have also presented recommendations and measurement methods in this issue. However, based on surveys there are only few web pages in the world that fully meet the proposed technical guidelines. Examining the top 10 visited webpages of the world, of the United States and of Hungary shows that our country is lagging behind using these new recommendations and methods.

Solution

In the third phase of my research I tested and applied methods proposed by literature and own experiences on a 4000 unique visitors a day web portal. Based on the results of performance analysis I created a method collection that is suitable not only for converting existing web pages, but can also be applied in design phase, thus greatly facilitate the births of faster intelligent web systems.

With the title *Improved Speed on Intelligent Web Sites* I published my related results in 2013, which arose the attention of **Egyptian** scientists, they cite my work in their *XML Schema-Based Minification for Communication of Security Information and Event Management (SIEM) Systems in Cloud Environments (DOI: 0.14569/IJACSA.2014.050912)* paper published in International Journal of Advanced Computer Science and Applications.

Implicit data collection, a new model

Problem

Using recommender systems is the main tool for implementing personalized web content. However, most of current research face a problem that it is very hard to gather real data in adequate quantity and quality, so the majority of scientists work on sample databases or the so fortunate minority of them can apply their data and text mining methods on real systems.

But even in the case of highly frequented and well known web systems like amazon.com or eBay.com it is very difficult to convince users to constantly fill out surveys, provide preference values. First of all, it is a time consuming task, on the other hand, users are wary, they tend to give out information about themselves less and less frequently.

Solution

That is why related researches are moving toward to implicit data collection. During implicit data collection web based systems can continuously collect and process large amount of data in an unobtrusive way. In this part of my research I provide a new, AJAX-based technology to make data acquisition more efficiently.

Although I published the results in my *AJAX-Based Data Collection Method for Recommender Systems* title article in 2012, the method described therein is still current, a trio of **Korean** researchers, Sung Moon Bae et al cited it in their *Utilization of Demographic Analysis with IMDB User Ratings on the Recommendation of Movies (DOI: 10.7838/jsebs.2014.19.3.125)* paper.

Responsive and content-aware presentation

Problem

Personalized web content is extremely important just as the fact that systems can be able to provide it in a user friendly way. The major drawback of earlier web systems is that they have not provided the so-called “application feeling” to the user, the method “we click and get the content immediately” which was familiar in desktop environment has not been prevailed due to the characteristics of a client-server architecture. The browser waits until it gets the answer for his request, so does the user as well. Each of us has spent several seconds in front of a computer waiting for web page loading or refresh. Developers intended to remedy this deficiency with the asynchronous communication capabilities of AJAX.

In addition, thanks to mobile device penetration and due to the introduction of Ipv6 as we can connect the Internet from our refrigerator or coffee maker as well, developers have to face a new challenge: it is not enough to develop content-dependent web systems, device-dependent webpages will own the future.

Solution

Accordingly, user interfaces developed with the mentioned model adapts intelligently to the device that display it, to its type and screen resolution. The information acquired about users is divided into three different groups: Device, Environment and Behaviour. These categories carry on the data collection based on the following sensor architecture.

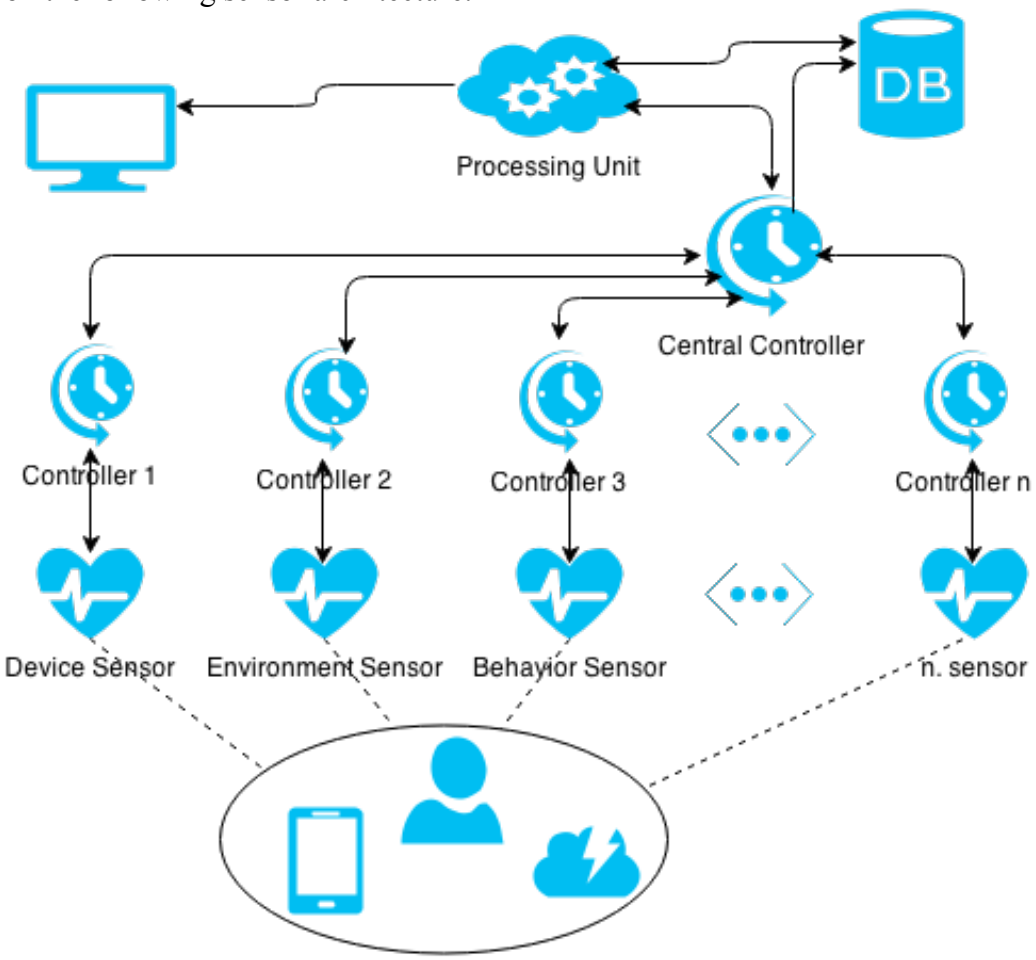


Fig. 3: Sensor architecture

Based on the information collected, the application server assemble the most appropriate content for the user, create the user interface and send it to the client device.

Privacy from forensic point of view

Problem

Right to privacy issues constantly arise related to intelligent web systems, because information security and personal data protection are key issues today. The invisible and continuous data collection has paramount importance not only of intelligent systems, but in terms of our entire life spent on the Internet.

As a forensic expert I worked in several criminal cases, where I had to discover the footprints of Internet usage on a given computer.

Solution

Experiences are surprising and enlightening. I had the pleasure to examine several aspects of the issue; so the point of view of the user, of the expert, of the criminal or of the investigating authority. The last phase of my research is focused on finding out the kind of information that has been collected and stored about a user by web-based systems, how and by what kind of tools, methods is it possible to identify and to defence against unauthorized hands.

I wrote two papers about results in this area in 2012. The first one was *Using Forensic Techniques for Internet Activity Reconstruction*, which was cited by **Chinese** researchers, Chen Long and others cited it in their *User browsing-data recovery of Google browser in private-browsing mode (DOI: 0.3979/j.issn.1673-825X.2013.06.027)* article, while the second one, *Social media risks from forensic point of view* was useful for Mohammad Reza Keyvanpour and his **Iranian** researcher colleagues in 2014. They cited it in their *Digital Forensics 2.0 (DOI: 10.1007/978-3-319-05885-6_2)* paper. It is a special pleasure that the latter article is mentioned by the web page of Information Systems Information Analysis Center (CSIAC) supported by the american Department of Defense.

Függelék

Nr.	Tevékenység	Javaslat				
		Yahoo	Google	HP WS	EF WS	Saját
1	Expires vagy Cache-Control Header hozzáadása	x		x	x	x
2	404s / hibás kérések elkerülése	x	x			
3	CSS @import mellőzése	x	x			x
4	CSS kifejezések mellőzése	x	x	x	x	
5	document.write mellőzése		x			
6	Üres IMG SRC mellőzése	x				x
7	Filterek (szűrők) mellőzése	x				
8	iframe-ek mellőzése, minimalizálása	x			x	
9	Átirányítások (redirect) mellőzése, csökkentése	x	x	x	x	x
10	Külső CSS-ek és JavaScriptek kombinálása		x			
11	Képek kombinálása CSS sprite-ok segítségével	x	x		x	
12	Etag-ek beállítása	x		x	x	
13	JavaScript-ek helyes sorrendben való használata		x		x	x
14	JavaScript-ek késleltetett feldolgozása		x			x
15	Okos eseménykezelők fejlesztése	x				
16	HTML-ben ne méretezzünk képeket	x	x			x
17	Az UI szálat ne blokkoljuk				x	
18	Gzip tömörítés engedélyezése	x	x	x	x	x
19	Mihamarabb jelenítsük meg a tartalmat	x			x	
20	Inline szkriptek a stíluslapok elé kerüljenek				x	
21	Az egyes komponensek 25 KB alattiak legyenek	x				
22	Használjunk böngésző cache-elést		x			x
23	Használjunk proxy cache-elést		x			
24	Asszinkron szkriptbetöltés				x	
25	Legyen az Ajax cache-elhető	x		x	x	
26	Legyen a favicon.ico kicsi és cache-elhető	x				
27	Kevesebb HTTP Request	x		x	x	x
28	A JavaScript és a CSS legyen külső	x		x	x	x
29	A landing page-ek legyenek cache-elve		x			
30	Minimalizáljuk a HTML-t		x			x
31	Minimalizáljuk a JavaScriptet és a CSS-t	x	x	x	x	x
32	Minimalizáljuk a DOM eléréseket	x				x
33	Minimalizáljuk a HTTP request méretet		x			x
34	Minimalizáljuk tömörítetlen tartalmat				x	x
35	Optimalizáljuk a képeket	x	x		x	x
36	Csomagoljuk a komponenseket egy Multipart dokumentumba	x				
37	Párhuzamosítsuk a letöltést több aldomain segítségével	x	x		x	
38	Késleltetett komponens betöltés	x				
39	Asszinkron erőforrások preferálása		x			
40	Előrehozott komponens betöltés	x				
41	A szkripteket a lap aljára	x	x	x	x	x
42	A stíluslapokat a lap tetejére	x	x	x	x	x
43	Csökkentsük a Cookie-k méretét	x				
44	Minimalizáljuk a DNS Lookup-okat	x	x	x	x	x
45	Csökkentsük a DOM elemek számát	x				x
46	Távolítsuk el a duplikált szkripteket	x		x	x	x
47	Távolítsuk el a nem használt CSS-t		x			x
48	Az erőforrásokat ugyanarról az URL-ről szolgáljuk ki		x			
49	Statikus tartalmat cookiementes domainről szolgáltassunk	x	x			
50	Egyszerűsítsük és használjunk a CSS szelektorokat		x		x	
51	Állítsunk be karakterkódolást		x			x
52	Adjunk meg képméretet		x			x
53	Használjunk Content Delivery Network (CDN) -öt	x		x	x	
54	Használjunk GET-et az Ajax kérésekhez	x				
55	Írjunk hatékony JavaScript-et				x	x

Ábrajegyzék

Irodalomjegyzék

1. DiNucci, Darcy: "Fragmented Future" (PDF). Print 53 (4): 32, 1999
2. O'Reilly T.: What Is Web 2.0, http://oreilly.com/pub/a/web2/archive/what-is-web-20.html, 2005
3. Berners-Lee T.: Plenary talk at WWWF94, http://www. w3.org/Talks/WWW94Tim/, 1994
4. Berners-Lee T., Hendler J., Lassila O.: The Semantic Web. Scientific American, 284(5):34–43, 2001
5. Zhong N., Liu J., Yao Y. Y., Ohsuga S.: Web intelligence (WI), Proc. 24th IEEE Computer Society International Computer Software and Applications Conference (COMPSAC 2000), Piscataway, NJ: IEEE CS Press, pp. 469–470, 2000
6. Liu J.: "Web Intelligence (WI): What Makes Wisdom Web?", in Proc. of the Eighteenth International Joint Conference on Artificial Intelligence, 1596-1602, 2003
7. Terziyan V.: Intelligent Web Applications, University of Jyvaskyla, 2002
8. Simon A. R., Shaffer S. L.: Data Warehousing and Business Intelligence for e-Commerce, San Francisco, CA: Morgan Kaufmann, 2001
9. Zhong N., Liu J., Yao Y.: "In Search of the Wisdom Web", in special issue on Web Intelligence (WI), IEEE Computer, Vol. 35, No. 11 (November 2002) 27-31.
10. Su Ho Ha: Helping Online Customers Decide through Web Personalization Intelligent Systems, IEEE (Vol. 17 , Issue: 6), 2002, pp. 34-43
11. Wang Y., Kobsa A.: Technical Solutions for Privacy-Enhanced Personalization, Intelligent User Interfaces: Adaptation and Personalization Systems and Technologies. Hershey, IGI Global, 2008
12. Su X., Khoshgoftaar T. M.: A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence, vol. 2009, Nr. 4, 2009.
13. Pazzani M. J., Billsus D.: Content-Based Recommendation Systems. The Adaptive Web, Lecture Notes In Computer Science, Vol. 4321, 2007, pp 325–341
14. Burke R.: Knowledge-Based Recommender Systems. Encyclopedia of Library and Information Science, 69 (32), 2000

15. Pazzani M. J.: A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review, Vol. 13, Nr. (5-6), 1999, pp. 393-408
16. Burke R.: Hybrid Recommender Systems: Survey and Experiments. User Modeling and User- Adapted Interaction, Vol. 12 Nr. 4, 2002, pp. 331– 370
17. Melville P., Mooney R. J.,Nagarajan R.: Content-Boosted Collaborative Filtering for Improved Recommendations Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002), pp. 187-192, Edmonton, Canada, July 2002
18. Sobecki J.: Implementations of Web-based Recommender Systems Using Hybrid Methods, International Journal of Computer Science & Applications Vol. 3 Issue 3, 2006, pp 52 – 64
19. Nagy Z.: Turisztikai portálok szolgáltatásai most és a jövőben, V. Nyíregyházi Doktorandusz (PhD/DLA) Konferencia Kiadványa, Nyíregyháza, 2011, pp. 98-102
20. Lütters H.: web 2.0 in the tourism industry – status quo, ITB Berlin, Berlin, 2010
21. A.L. Barabási: Behálózva. A hálózatok tudománya, Budapest, Helikon Kiadó, 2013
22. Nagy Z.: Improved Speed on Intelligent Web Sites, Recent Advances in Computer Science, Rhodes Island, Greece, 2013, pp. 215-220
23. Powell T. A.: Web design: The complete guide. New York: McGraw-Hill, 2000.
24. Murugesan S. Ginige A.: Web Engineering: Introduction and Perspectives. Software Engineering, 1999
25. Deshpande Y., et al.: Web engineering. Journal of Web Engineering, 1(1), 3-17, 2002
26. Deshpande Y., Hansen S.: Web Engineering: Creating a Discipline among Disciplines, IEEE Multimedia, Special issues on Web Engineering, vol 8, no 2, pp 82-87, 2001
27. Pressman R. S.: Web Engineering: An Adult's Guide to Developing Internet-Based Applications, Cutter IT Journal, vol 14, no. 7, pp 2-5, 2001
28. Ginige A., Murugesan S.: The essence of Web engineering: Managing the diversity and complexity of Web application development. IEEE Multimedia, 8(2), 22-25., 2001
29. Royce W. W.: Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26 (August): 1–9, 1970

30. Biró M.: The Software Process Improvement Hype Cycle, UPGRADE - CEPIS 10: (5) pp. 14-20.,2009
31. Boehm B.: A Spiral Model of Software Development and Enhancement, IEEE Computer, IEEE, 21(5):61-72, 1988
32. Gilb T.: Principles of Software Engineering Management, Addison-Wesley, 1988
33. Adamkó A.: Web Alapú Információs Rendszerek Modellezése, Doktori Értekezés, Debreceni Egyetem, 2008
34. Ginige A.,Murugesan S.: Web engineering: A methodology for developing scalable, maintainable Web applications. Cutter IT Journal, 14(7), 24-35.,2001
35. Viljami S.: Responsive Workflow, http://viljamis.com/blog/2012/responsive-workflow/, 2012
36. Boulton M.: Responsive Summit Workflow, http://www.markboulton.co.uk/journal/responsive-summit-workflow, 2012
37. UXMovement: Why it is important to sketch before you wireframe, http://uxmovement.com/wireframes/why-its-important-to-sketch-before-you-wireframe/, 2012
38. Blodget H., Danova T.: The Future of Digital:2013, http://www.businessinsider.com/the-future-of-digital-2013-2013-11?op=1, Business Insider, 2013
39. Reenskaug T.: The Original MVC Reports 1979, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html, 2007
40. Apple Developer: Cocoa Core Competencies: Model-View-Controller, https://developer.apple.com/library/mac/documentation/general/conceptual/devpedia-cocoacore/MVC.html, Szeptember 2013
41. Potel M.: MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. Taligent Inc., 1996
42. Greer D.: Interactive application architecture patterns. http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/, 2007
43. Bower A., McGlashan B.: Twisting the triad: The evolution of the dolphin smalltalk

mvp application framework. Tutorial Paper for ESUG, 2000.

44. Fowler M.: Presentation model http://martinfowler.com/eaaDev/PresentationModel.html, 2004
45. Smith J.: WPF Apps With The Model-View-ViewModel Design Pattern, http://msdn.microsoft.com/en-us/magazine/dd419663.aspx, 2009
46. Microsoft Developer Network: Implementing the Model-View-ViewModel Pattern, http://msdn.microsoft.com/en-us/library/ff798384.aspx
47. Smith J.: Using MVC to Unit Test WPF Applications, http://www.codeproject.com/Articles/23241/Using-MVC-to-Unit-Test-WPF-Applications, 2008
48. Rossi G., Schwabe D., Guimarães R.: Designing personalized web applications. In Proceedings of the 10th international conference on World Wide Web, ACM, 2001, pp. 275-284.
49. Pazzani M. J., Billsus D.: Content-Based Recommendation Systems. The Adaptive Web, Lecture Notes In Computer Science, Vol. 4321, 2007, pp 325–341
50. Su X., Khoshgoftaar T. M.: A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence, vol. 2009, Nr. 4, 2009.
51. Dell'Aglio D., Celino I., Cerizza D.: Anatomy of a Semantic Web-enabled Knowledge-based Recommender System, http://www.larkc.eu, 2010
52. Marcotte E.: Responsive web design. A List Apart, 306., 2010, http://alistapart.com/article/responsive-web-design
53. StrangeLoop Networks: Website abandonment happens after 3 seconds, http://www.strangeloopnetworks.com/resources/info graphics/web-performance-and-user- expectations/website-abandonment-happens-after-3- seconds/, 2010
54. Lohr S.: Impatient web users flee slow loading sites, New York Times, 2012 http://www.nytimes.com/2012/03/01/technology/im patient-web-users-flee-slow-loading-sites.html
55. Kissmetrics.com: Loading time, http://blog.kissmetrics.com/loading-time/, 2013
56. Ecoconsultancy.com: Slow loading websites cost retailers 1,73 bn in lost sales each year, http://econsultancy.com/hu/blog/9790-slow-loading- websites-cost-retailers-1-73bn-in-lost-sales-each- year, 2013

57. StrangeLoop Networks: Internet users have faulty perceptions of time, ttp://www.strangeloopnetworks.com/resources/infog raphics/web-performance-and-user- expectations/internet-users-have-faulty-perceptions- of-time/ , 2013
58. Langville N., Meyer Carl D.: Deeper Inside PageRank, Internet Mathematics, Vol 1, No. 3, pp. 335-400, 2004
59. Econsultancy: It's official: Google incorporates website speed into your ranking, http://econsultancy.com/hu/blog/5728-it-s-official-google-incorporates-website-speed-into-your-ranking, 2013
60. Google Developer: Make the web faster, https://developers.google.com/speed/, 2013
61. Yahoo: Exceptional Performance, http://developer.yahoo.com/performance/, 2013
62. Souders S.: High Performance Web Sites. O'Reilly Media, 2007
63. Souders S.: Even faster web sites: performance best practices for web developers. O'Reilly Media, 2009
64. IETF: Hypertext Transfer Protocol version 2.0, http://tools.ietf.org/html/draft-ietf-httpbis-http2-07, 2013
65. Willis N.: Reducing HTTP latency with SPDY, http://lwn.net/Articles/362473/,2009
66. IETF: Hypertext Transfer Protocol (HTTP) Keep-Alive Header, http://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html, 2012
67. Sixrevisions.com: Free website speed testing, http://sixrevisions.com/tools/free-website-speed- testing/, 2013
68. StatCounter: http://gs.statcounter.com/#resolution-ww-monthly-201306-201308-bar, 2013 augusztus
69. Ramachandran S.: Web metrics: Size and number of resources, https://developers.google.com/speed/articles/web-metrics, 2013
70. Akami Technologies: The State of the Internet, 2013, vol 6. num. 1 http://www.akamai.com/dl/akamai/akamai_soti_q113.pdf?WT.mc_id=soti_Q113
71. KSH: Statisztikai Tükör Távközlés, internet, kábeltelevízió, 2012, http://www.ksh.hu/docs/hun/xftp/idoszaki/tavkint/tavkint12.pdf
72. KSH: Statisztikai Tükör Távközlés, internet, kábeltelevízió, 2008, http://www.ksh.hu/docs/hun/xftp/idoszaki/tavkint/tavkint08.pdf
73. KSH: Statisztikai Tükör Távközlés, internet, kábeltelevízió, 2010,

http://www.ksh.hu/docs/hun/xftp/idoszaki/tavkint/tavkint10.pdf

74. Techyfuzz.com: http://techyfuzz.com/free-cdn- content-delivery-network-services-website/, 2013
75. Wilde E., Gaedke M.: Web Engineering Revisited. In: BCS Int. Acad. Conf. 2008. p. 41-50.
76. Allaire J.: Macromedia Flash MX - A next-generation rich client. http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf, 2002
77. Bozzon A., Comai S., Fraternali P., Carughi G. T.: Capturing RIA concepts in a web modeling language. In Proceedings of the 15th international Conference on World Wide Web WWW '06. ACM, New York, 907-908, 2006
78. Meliá S., Gómez J., Pérez S., Díaz O.: A Model-Driven Development for GWT-based Rich Internet Applications with OOH4RIA. Proc. 8th Int. Conf. on Web Engineering (ICWE'08). IEEE, pp.13 – 23, 2008.
79. Deitel P. J., Deitel H. M.: AJAX, Rich Internet Applications, and Web Development for Programmers, Prentice Hall International, 2008
80. Busch M., Koch N.: Rich Internet Applications. Technical Report 0902, Institute for Informatics, Ludwig-Maximilians-Universität München, 2009.
81. Garrett J. J. : Ajax: A New Approach to Web Applications: http://www.adaptivepath.com, 2005
82. Costello E.: Remote Scripting with IFRAME, http://www.oreillynet.com/pub/a/javascript/2002/02/ 08/iframe.html, 2002
83. Nagy Z.: Ajax 2 in 1: Interactive education and modern web technology, Journal of Applied Multimedia, vol.: 1, num.: 1, 2010, pp. 39-44
84. W3Trends: Usage of Javascript libraries for websites, http://w3techs.com/technologies/overview/javascript_library/all, 2013
85. Pingdom: The Web loves jQuery, and here are the numbers to prove it, http://royal.pingdom.com/2012/06/20/jquery-numbers/, 2012

86. W3Trends.com: Usage of server-side programming languages for websites, http://w3techs.com/technologies/overview/programming_language/all, 2014 január
87. WebTrends: Usage of content management systems for websites, http://w3techs.com/technologies/overview/content_management/all, 2014 január
88. Google Developers: Building Smartphone-Optimized Websites, https://developers.google.com/webmasters/smartphone-sites/details, 2013 december
89. W3C: Media Queries, http://www.w3.org/TR/css3-mediaqueries/, 2012 június
90. Nagy Z.: Social media risks from forensic point of view, International Journal of Computers and Communications 6 (4), 245-253, 2012
91. LiveCareer.com: Your Facebook Profile Could Affect Your Hiring Potential, http://www.livecareer.com/press-releases/your-facebook-profile-could-affect-your-hiring-potential, 2012 május

Publikációs lista / Publications

Nagy Zsolt közleményei

2014

1 Zsolt Nagy
Adaptive Design Process for Responsive Web Development
In: Nikos Mastorakis, Kleanthis Psarris, Kleanthis Psarris, Kleanthis Psarris, Valeri Mladenov, Aida Bulucea, Imre Rudas, Olga Martin (szerk.)
Advances in Information Science and Applications: Proceedings of the 18th International Conference on Computers (part of CSCC '14). Konferencia helye, ideje: Santorini, Görögország, 2014.07.17-2014.07.21. Bulgaria: European Society for Applied Sciences and Development, 2014. pp. 410-414.
(ISBN:978-1-61804-237-8)
Link(ek): DOI, ResearchGate publ., Kiadónál
Könyvrészlet/Konferenciaközlemény/Tudományos [2727387]
[Admin láttamozott]

2013

2 Zsolt Nagy
Improved Speed on Intelligent Web Sites
In: Nakov Ognian, Borovska Plamenka, Antonio Abelha (szerk.)
Recent Advances in Computer Science: Proceedings of the 17th International Conference on Computers. Konferencia helye, ideje: Rhodes, Görögország, 2013.07.16-2013.07.19. Bulgaria: WSEAS Press, 2013. pp. 215-220.
(ISBN:978-960-474-311-7)
Link(ek): Kiadónál, ResearchGate publ.
Könyvrészlet/Konferenciaközlemény/Tudományos [2429481]
[Admin láttamozott]
Független idéző: 1 Összesen: 1
1 *Bishoy Moussa, Mahmoud Mostafa, Mahmoud El-Khouly*
XML Schema-Based Minification for Communication of Security Information and Event Management (SIEM) Systems in Cloud Environments
INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS (ISSN: 2158-107X) 5: (9) *pp. 74-82. (2014)*
Link(ek): DOI
Folyóiratcikk/Szakcikk/Tudományos [14227424]

2012

3 Nagy Zsolt
Turisztikai portálok szolgáltatásai most és a jövőben
In: Kerekes Benedek, Gát György (szerk.)
V. Nyíregyházi Doktorandusz (PhD/DLA) Konferencia Előadásainak közleménye. Konferencia helye, ideje: Nyíregyháza, Magyarország, 2011.12.07 Nyíregyháza: Nyíregyházi Főiskola Tudományos Tanácsa, 2012. pp. 98-102.
(ISBN:978-963-9909-9)
Könyvrészlet/Konferenciaközlemény/Tudományos [2328324]
[Admin láttamozott]

4 Zsolt Nagy
AJAX-Based Data Collection Method for Recommender Systems
In: Sandra Sendra, José Carlos Metrolho, Vladislav Skorpil, Antoanela Naaji (szerk.)
Recent Researches in Communications and Computers: Proceedings of the 16th WSEAS International Conference on Computers (part of CSCC '12). Konferencia helye, ideje: Kos, Görögország, 2012.07.14-2012.07.17. Sofia: World Scientific and Engineering Society Press, 2012. pp. 446-451.
(ISBN:978-1-61804-109-8)
Link(ek): ResearchGate publ.
Könyvrészlet/Konferenciaközlemény/Tudományos [2328241]
[Admin láttamozott]
Független idéző: 1 Összesen: 1
1 *Sung Moon Bae, Sang Chun Lee, Jong Hun Park*
Utilization of Demographic Analysis with IMDB User Ratings on the Recommendation of Movies
The Journal of Society for e-Business Studies (ISSN: 2288-3908) 19: (3) *pp. 125-141. (2014)*
Link(ek): DOI, Kiadónál
Folyóiratcikk/Szakcikk/Tudományos [14170079]

5 Zsolt Nagy
Using Forensic Techniques for Internet Activity Reconstruction
In: Sandra Sendra, José Carlos Metrolho, Vladislav Skorpil, Antoanela Naaji (szerk.)
Recent Researches in Communications and Computers: Proceedings of the 16th WSEAS International Conference on Computers (part of CSCC '12). Konferencia helye, ideje: Kos, Görögország, 2012.07.14-2012.07.17. Sofia: World Scientific and Engineering Society Press, 2012. pp. 248-253.
(ISBN:978-1-61804-109-8)

Link(ek): ResearchGate publ., Google scholar
Könyvrészlet/Konferenciaközlemény/Tudományos [2328244]
[Admin láttamozott]
Független idéző: 1 Összesen: 1

1 *Chen Long, Zhang Lei, Tian Qingyi*
User browsing-data recovery of Google browser in private-browsing mode
Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition) (ISSN: 1673-825x) 25: (6) *pp. 854-858. (2013)*
Link(ek): DOI
Folyóiratcikk/Szakcikk/Tudományos [14067740]

6 Zsolt Nagy
Social media risks from forensic point of view
INTERNATIONAL JOURNAL OF COMPUTERS AND COMMUNICATIONS 6:(4) pp. 245-253. (2012)
Link(ek): ResearchGate publ., Teljes dokumentum, Google scholar
Folyóiratcikk/Szakcikk/Tudományos [2328321]
[Admin láttamozott]
Független idéző: 1 Összesen: 1

1 *MohammadReza Keyvanpour, Mohammad Moradi, Faranak Hasanzadeh*
Digital Forensics 2.0: A Review on Social Networks Forensics
In: Azah Kamilah Muda, Yun-Huoy Choo, Ajith Abraham, Sargur N Srihari (szerk.) : Computational Intelligence in Digital Forensics : Forensic Investigation and Applications. (555) New York [etc.]: Springer, 2014. (ISBN 978-3-319-05884-9) *pp. 17-46*. (Studies in Computational Intelligence)
Link(ek): DOI
Könyvrészlet/Szaktanulmány/Tudományos [14067787]

2011

7 Nagy Zs.
Intelligent web System and its life essence: the AJAX
In: Attila Egri-Nagy, Emőd Kovács, Gergely Kovásznai, Gábor Kusper, Tibor Tómács (szerk.)
Proceedings of the 8th International Conference on Applied Informatics (ICAI2010). Konferencia helye, ideje: Eger, Magyarország, 2010.01.27-2010.01.30. Eger: Eszterházy Károly Főiskola, 2011. pp. 351-357.
I-II.
(ISBN:978-963-9894-72-3)
Link(ek): ResearchGate publ., Teljes dokumentum
Befoglaló mű link(ek): Teljes dokumentum, BME PA közlemény
Könyvrészlet/Konferenciaközlemény/Tudományos [2328265]
[Admin láttamozott]

2010

8 Nagy Zs, Pusztai Zs, Gönczi P.
DMK – www.debrecenimuvkozpont.hu

(2010)
Link(ek): Egyéb URL
Egyéb/Számítógépes program/Tudományos [2328621]
[Admin láttamozott]

9 Nagy Zsolt
Intelligent Web Systems
In: Vincze Krisztián (szerk.)
A Nemzetközi III. Nyíregyházi Doktorandusz (PhD/DLA) Konferencia Kiadványa. Konferencia helye, ideje: Nyíregyháza, Magyarország, 2009.11.20 Nyíregyháza: Szent Atanáz Görög Katolikus Hittudományi Főiskola, 2010. pp. 50-54.
(ISBN:978 963 87809 6 6)
Könyvrészlet/Konferenciaközlemény/Tudományos [2328316]
[Admin láttamozott]

10 Nagy Zsolt
Web 2.0, Az Intelligens Web
In: Cserny László (szerk.)
INFORMATIKA KORSZERŰ TECHNIKÁI KONFERENCIA 2010. Konferencia helye, ideje: Dunaújváros, Magyarország, 2010.03.05-2010.03.06. Dunaújváros: Dunaújvárosi Főiskola, 2010. pp. 165-170.
(ISBN:978-963-9915-38-1)
Könyvrészlet/Konferenciaközlemény/Tudományos [2328328]
[Admin láttamozott]

11 Nagy Zsolt
Öntanuló webrendszerek
In: Networkshop 2010 : 19. Országos Konferencia: kivonatok=abstracts. Konferencia helye, ideje: Debrecen, Magyarország, 2010.04.07-2010.04.09. Budapest: Nemzeti Információs Infrastruktúra Fejlesztési Intézet, Paper 148.
Link(ek): Teljes dokumentum
Egyéb konferenciaközlemény/Konferenciaközlemény/Tudományos [2328359]

[Admin láttamozott]

12 Zsolt Nagy
Ajax 2 in 1: interactive education and modern web technology
JOURNAL OF APPLIED MULTIMEDIA 5:(1) pp. 39-44. (2010)
Link(ek): ResearchGate publ., Google scholar
Folyóiratcikk/Szakcikk/Tudományos [2328179]
[Admin láttamozott]

2009

13 Nagy Zs, Pusztai Zs
Kölcsey Televízió – www.kolcseytv.hu

(2009)
Link(ek): Egyéb URL
Egyéb/Számítógépes program/Tudományos [2328622]
[Admin láttamozott]

2008

14 Nagy Zs, Pusztai Zs
Hotel Lycium – www.hotellycium.hu

(2008)
Link(ek): Egyéb URL
Egyéb/Számítógépes program/Tudományos [2328611]
[Admin láttamozott]

2007

15 Nagy Zs, Pusztai Zs
Aquaticum.hu

http://www.aquaticum.hu/ (2007)
Link(ek): Egyéb URL
Egyéb/Számítógépes program/Tudományos [2328593]
[Admin láttamozott]

2004

16 Nagy Zs, Pusztai Zs
BelföldiSzállások.hu

http://www.belfoldiszallasok.hu/ (2004)
Link(ek): Egyéb URL
Egyéb/Számítógépes program/Tudományos [2328495]
[Admin láttamozott]