Viktors Krasovskis
Daugavpils Universitāte, Latvija

DEKLARATĪVĀS PROGRAMMĒŠANAS PIELIETOŠANA TĪMEKĻA LIETOTŅU GRAFISKĀS SASKARNES IZVEIDĒ

Abstract

Appliance of declarative programming in web application GUI development

Nowadays it is a trend to simplify Web application development process both in server logic and GUI level. Lack of runtime validation and object-oriented GUI interpretation cause a need to replace HTML, XHTML or similar GUI definition languages with a more simple and transparent solution. Recently simple declarative human readable data exchange formats such JSON, YAML became very popular. The goal of this article is to prove the indirect use of declarative data exchange formats for Web application GUI development.

The runtime HTML/XHTML validation is a wide spread problem, because many web applications use HTML/XHTML as strings in their GUI. HTML/XHTML code validation in IDE does not solve the problem fully because not all IDEs do support this functionality. To develop a framework level server side GUI validation system is very complicated because it involves wide use of regular expressions. So a more solid GUI definition and interpretation solution is needed.

Using MVC pattern the application is being split into 3 parts (data, logic, presentation), but in most modern web application framework use cases the view, which is being used for data presentation, could be further decomposed in two separate parts. The problem is that such web application frameworks like Rails or ASP.NET MVC use programming language embedding in the HTML/XHTML code. This means that two paradigms (imperative and declarative) are being mixed at one place. It is possible to develop views, which contain two parts. The first part uses declarative data exchange format (for example, JSON or XML), which defines the GUI, but the second part contains code, which turns to certain data and fills the GUI.

This article reviews possibility to interpret GUI on server side using an object tree like in the DOM (document object model) in a web browser. The GUI tree is being built from previously mentioned declarative GUI. This approach solves many problems, like runtime HTML/XHTML validation and very easy access to GUI elements.

Atslēgas vārdi: JSON, YAML, XML, HTML, XHTML, web, GUI, MVC

Ievads

Mūsdienās izceļas tendence maksimāli vienkāršot tīmekļa lietotņu izstrādes procesu. Daudzi izstrādātāji aktīvi pielieto tīmekļa lietotņu ietvarus, kuri dod iespēju sadalīt lietotnes struktūru 3 neatkarīgās daļās: datos, loģikā un datu pasniegšanā (Reenskaug). Tīmekļa lietotņu izstrāde var līdzināties darbvirsmas lietotņu izstrādei, tomēr grafiskās saskarnes ziņa ir krasas atšķirības (Shan, Tony 2006). Galvenās tīmekļa lietotņu grafisko saskarņu problēmas ir

1. konkrētu izpildlaika validācijas mehānismu trūkums;
2. objektorientēta grafiskās saskarnes interpretācija servera pusē;
3. stāvokļa saglabāšana.

Šajā rakstā tiks apskatīt iespējamie doto problēmu risinājumi, pielietojot deklaratīvus datu apmaiņas formātus saskarnes definēšanai.

Grafiskās saskarnes validācijas problēma

Lielākā daļa pārlūkprogrammu māk attēlot tekstveida HTML un XHTML formāta datus, ar kuru palīdzību tiek definēta grafiskā saskarne. Atkarībā no pārlūkprogrammas

izkārtojuma dzinēja, var tikt kļūdaini attēloti nekorekti HTML vai XHTML dati, nebrīdinot lietotāju. Līdz ar to izpildlaika grafiskās saskarnes validācija ir ļoti aktuāla problēma (Jacobs, Walsh). Protams, pastāv integrētās tīmekļa izstrādes vides, kuras nodrošina, piemēram, HTML koda validāciju. Tādā veidā, izstrādājot tīmekļa lietotnes, var izvairīties no nekorekta saskarnes koda. Dotā pieeja nerisina problēmu galīgi, jo pastāv izvēles brīvība un izstrādātājs iespēju robežās var izvēlēties sev ērtāko izstrādes vidi, kurā ir nepieciešamās iespējas, bet var arī nebūt HTML/XHTML koda validācijas. Daudzos tīmekļa izstrādes ietvaros pastāv grafiskās saskarnes mehānismi - skatu dzinēji (*view engine*), piemēram, ASP.NET MVC *Razor* vai *Rails* ERB, kuri nodrošina programmēšanas valodas un HTML/XHTML savietojamību (skat. Attēls 1). Šāds HTML/XHTML kods satur daļas, kuras bez iepriekšējās apstrādes tiek nodotas pārlūkprogrammai un daļas, kuras tiek izpildītas serverī. Arī tāda formāta kods ir grūti validējams un jauc kopā 2 atšķirīgas paradigmas - deklaratīvu un imperatīvu. Tāpēc visoptimālākais risinājums būtu grafiskās saskarnes mehānismu iekļaušana tīmekļa lietotnes izstrādes ietvarā. Tomēr tas ir grūti izdarāms, jo būs jāpielieto sarežģītas regulāras izteiksmes tekstveida datu validācijai (Thereaux).

```
<div id='wrapper'>
  <div id='sidebar'>
    <%= yield :sidebar %>
  </div>
  <div id='content'>
    <%= yield %>
  </div>
</div>
```

Attēls 1. Iegultais *Ruby* kods.

Grafiskās saskarnes objektorientēta interpretācija

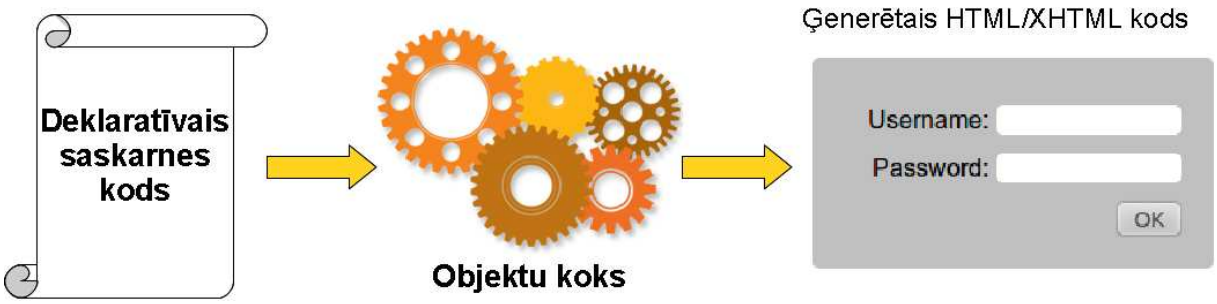
Ja līdz šim brīdim MVC projektēšanas šablona galvenais uzdevums bija sadalīt lietotnes struktūru 3 neatkarīgās daļās (modelis, skats, kontrolieris), tad nākamais loģiskais solis būtu kādas daļas tālākā dekompozīcija. Kā jau iepriekš minēts, daudzu tīmekļa lietotņu ietvaru izpratnē skats apvieno gan HTML/XHTML kodu, gan ietvara programmēšanas valodu. Tāda skatu interpretācija slikti ietekmē uz to atkārtotu pielietošanu, jo tie satur kodu, kurš vēršas pie konkrētiem datiem. Ideoloģiski korektāk būtu skata interpretācija, kura sastāv no 2 komponentēm. Pirmā komponente definē tikai skata ārējo izskatu un nesatur scenāriju, bet otrā komponente nodrošina vēršanos pie konkrētiem datiem un pirmās komponentes piepildīšanu ar tiem. Tā kā pirmā skata komponente nesatur loģiku, ciklus un citas darbības, to ir iespējams definēt, pielietojot kādu deklaratīvu datu formātu. Pēc DOM (*document object model*) ideoloģijas, katrs tīmekļa lapas elements ir objekts, kuram piemīt konkrētas īpašības, tā notikumu apstrādātāji, kā arī tas var sevī saturēt citus DOM elementus, veidojot objektu koku. Tīmekļa pārlūkā var darboties ar DOM elementiem izmantojot *JavaScript*

objektorientēto pieeju, jo pārlūkprogrammas izkārtojuma dzinējs interpretē no servera saņemto HTML/XHTML kodu un konstruē attiecīgo objektu koku (Le Hégaret). Savukārt daudzās tīmekļa lietotnēs servera pusē skata daļas tiek interpretētas teksta veidā un ir salīdzinoši statiskas. Šī nianse arī rada būtisku atšķirību starp tīmekļa un darbvirsmas lietotnēm. Programmētājiem būtu daudz vienkāršāk darboties ar skatiem, ja tie tiktu interpretēti objektu koka veidā, kā tas ir ierasts darbvirsmas vai pārlūku *JavaScript* lietotņu izstrādē (Köhler).

Skatus varētu definēt gan ar HTML, gan ar XHTML, gan kādā citā formāts, kurš var glabāt datu struktūras teksta veidā, ir rediģējams un viegli uztverams cilvēkam. Piemēram, XML, JSON un YAML formāti nodrošina dažādu datu struktūru glabāšanu teksta veidā, kurus ir iespējams deserializēt un tīmekļa lietotņu ietvara līmenī servera pusē konstruēt skatus kā objektu kokus (skat. Attēls 2). Dotais mehānisms automātiski iekļauj ieejas datu validāciju. Objektorientēta skatu interpretācija tīmekļa lietotnē uz MVC pamata dod sekojošas priekšrocības:

- Vienkāršāka skatu testēšana
- Skatu un to daļu atkārtota izmantošana, piemēram, pielietojot mantošanu, agregāciju vai kompozīciju
- Vienkāršāka skatu projektēšana, lietojot UML klašu diagrammas

Jāņem vērā, ka objektu koku pielietošana MVC skatos nodrošina maksimālu abstrakcijas līmeni. Piemēram, programmētājam nav nepieciešamas specifiskas HTML/XHTML zināšanas, ja attiecīgā koda ģenerēšana tiks nodrošināta tīmekļa lietotnes ietvara līmenī. Līdz ar to tiek atrisināta izpildlaika skatu validācijas problēma, jo vienmēr tiek ģenerēts korekts HTML/XHTML kods.



Attēls 2. Deklaratīvā grafiskās saskarnes koda pielietošana.

Grafiskās saskarnes stāvokļa saglabāšana

Daudzas tīmekļa lietotnes darbojas uz notikumu principa, kuri tiek izpildīti, vēršoties pie konkrētiem URL (*uniform resource identifier*) (MSDN). Atkārtoti izsaucot vienu un to pašu URL, piemēram, aizpildot kādu formu tīmekļa lietotnes lapā, ir jāsaglabā starprezultāti un grafiskās saskarnes izskats. Parasti šāda veida operāciju algoritmus atsevišķi izstrādā

programmētājs un tas aizņem vairāk laika, kā arī pieaug kļūdu esamības varbūtība (Berners-Lee).

Tā kā ir iespējams lietot grafisko saskarni objektu koka veidā, ir iespējams to serializēt un saglabāt pēc katras darbības, ko izsauc konkrēts URL. Atkārtoti vēršoties pie dotā URL var deserializēt grafiskās saskarnes objektu koku, kurā tika saglabātas visas izmaiņas. Tiek veidot ilūzija, ka tiek izstrādāta darbvirsmas lietotne, kur grafiskā lietotne vai tās elementi ir pastāvīgi. Dotā pieeja ļauj programmētājam risināt uzdevumu, nenovēršot uzmanību uz otršķirīgām problēmām.

Secinājumi

Deklaratīvo datu apmaiņas formātu pielietošana tīmekļa lietotņu grafiskās saskarnes definēšanā ļauj veiksmīgi risināt iepriekš minētās problēmas. Deklaratīvi definētas datu struktūras ir viegli deserializējamas servera puses objektu kokā – būtībā tādā veidā arī tiek interpretēta grafiskā saskarne tīmekļa pārlūkā. Tādu deklaratīvu datu apmaiņas formātu kā XML, JSON, YAML interpretēšana ir iebūvēta daudzās programmēšanas valodās vai to standarta bibliotēkās, un līdz ar to ieejas datu validācija tiek maksimāli vienkāršota.

Izpētot mūsdienu populārākos tīmekļa lietotņu ietvarus, to problēmas un tendences, ver secināt sekojošo:

- Daudzu tīmekļa lietotņu tekstveida grafiskās saskarnes komponentes servera pusē ir novecojis risinājums, kas slikti ietekmē uz lietotnes kopējo integritāti;
- Pastāv varbūtība, ka nākamās paaudzes tīmekļa lietotņu ietvari pielietos galvenokārt objektorientētu pieeju darbam ar grafisko saskarni;
- Objektorientētās grafiskās saskarnes definēšanai var pielietot mūsdienīgus datu apmaiņas formātus ar lakonisku sintaksi, piemēram, JSON vai YAML.

Bibliogrāfija

1. Berners-Lee, T. Design Note: Linked Data. http://www.w3.org/DesignIssues/LinkedData.html [2011.05.05].
2. Jacobs, Walsh. "URI/Resource Relationships". Architecture of the World Wide Web, Volume One. http://www.w3.org/TR/webarch/ [2011.05.05].
3. Köhler, R. The OOP Approach on MVC UI - System.Web.UI.Controls. http://www.codeproject.com/KB/aspnet/OOP_on_MVC_UI.aspx [2011.05.05].
4. Le Hégaret, P. Document Object Model (DOM) Specifications. http://www.w3.org/DOM/DOMTR [2011.05.05].
5. MSDN. ASP.NET Routing. http://msdn.microsoft.com/en-us/library/cc668201.aspx [2011.05.05].
6. Reenskaug, T. MVC XEROX PARC 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html [2011.05.05].

7. Shan, Tony (2006) Taxonomy of Java Web Application Frameworks. Proceedings of 2006 IEEE International Conference on e-Business Engineering , ICEBE 2011.05.05].
8. Thereaux, O. The W3C QA Toolbox - Validators, checkers and other tools for Webmasters and Web Developers. http://www.w3.org/QA/Tools/ [2011.05.05].