

Smart Extension Framework

Autor:

Victor Hurdugaci
victor.hurdugaci@yahoo.com

Prof. Coordonator:

Lect. Dr. Lucian Sasu

Smart Extension Framework adaugă support pentru expansiune și modularizare a aplicațiilor. Integrarea lui cu aplicațiile se realizează foarte ușor și în general nu necesită decât 10-20 linii de cod. Diferența față de framework-urile existente este că SEF nu necesită abstract factory în plugin-uri ci doar o frază scrisă în limbaj apropiat de limbajul natural iar plugin-urile pot comunica unul cu celălalt.

Ce este un Framework?

În 21 Februarie 1997, Ralph E. Johnson, publica un articol intitulat "Components, Frameworks, Patterns" în care susținea că "Framework = Components + Patterns". Acesta este doar una din posibilele definiții ale unui framework dar este cât se poate de corectă deoarece componentele reprezintă cod și implementare iar tiparele reprezintă design pur, moduri de rezolvare a unor situații dar nu oferă cod. Un framework conține atât cod cât și tipare.

De asemenea, un framework poate fi asemănat cu un schelet, o temelie, pentru aplicație. Pentru a se observa cât de grea este definirea framework-ului, mai jos sunt 3 definiții posibile, toate trei acceptate și corecte:

- "A framework is the skeleton of an application that can be customized by an application developer"
- "Frameworks are an object-oriented reuse technique that are widely used in industry but not discussed much by the software engineering research community. They are a way of reusing design that is part of the reason that some object-oriented developers are so productive."
- "A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact."

Framework-urile își expun funcționalitatea prin API-uri și funcționează pe baza principului holywood-ian: "Don't call us, we'll call you".

Ce este Smart Extension Framework?

De multe ori se întâlnesc situații când se dorește o facilitate inexistentă într-o aplicație sau se dorește eliminarea alteia din diverse motive: consum mare de resurse, lipsa utilității, înlocuirea cu alta etc. Aplicațiile în general nu oferă suport pentru modularizare și de aceea lucrurile descrise mai sus sunt imposibile pentru un utilizator normal sau aproape imposibile pentru un utilizator avansat care deține cunoștințe de programare.

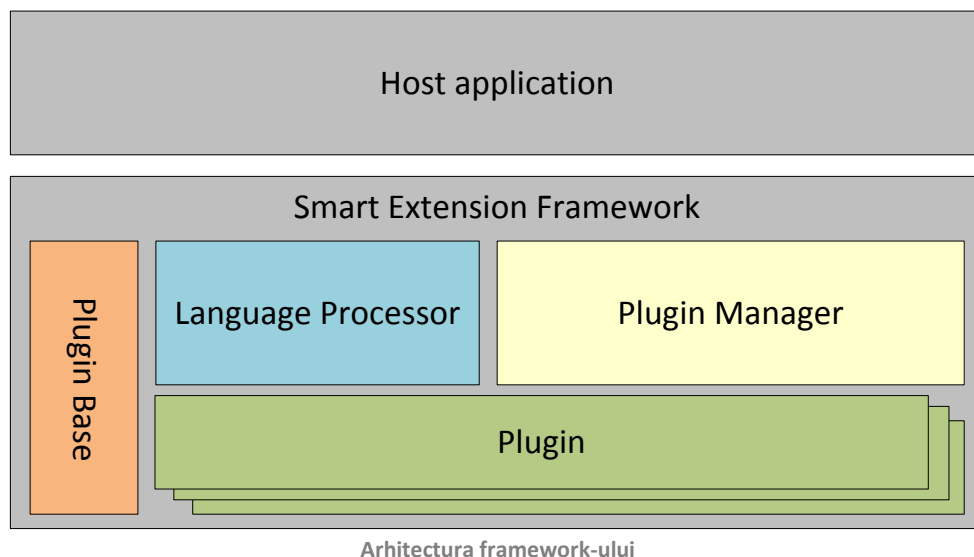
Smart Extension Framework vine în ajutorul programatorului oferindu-i o platformă, un suport pentru a adăuga extensibilitate în aplicațiile scrise de el. Acest lucru se realizează foarte ușor, nu necesită învățarea unui nou limbaj, nu necesită înțelegerea modului de lucru al SEF nici din partea celui care scrie aplicația gazdă nici a celor care scriu pluginuri. Tocmai de aceea, această platformă se

intitulează “Smart” – ea se ocupă de integrarea oricărui plugin, în orice aplicație fără ca utilizatorul sau dezvoltatorul să se preocupe de interoperabilitate.

Funcțiile expuse de un plugin sunt decorate (descrise) cu un atribut care conține o frază în limbaj natural. În versiunea aceasta (alpha) limbajul poate fi doar unul regulat (de tip 3) dar în versiunea finală acesta va putea fi limbaj naturaj. Acest lucru diferențiază Smart Extension Framework de alte framework-uri: pluginurile își descriu comportamentul urmând ca SEF să interogheze aceste fraze și să determine care succesiune de apeluri de metode duc la rezultat (dacă acesta exista).

Arhitectura

Din punct de vedere arhitectural, Smart Extension Framework este compus din 3 componente Plugin Manager, Language Processor si Plugin Base. Toate acestea ajută la încărcarea și utilizarea pluginurilor.



Componenta centrală, deși în diagramă nu este amplasată în centru, este Plugin Manager. Aceasta mediatizează comunicarea dintre aplicația gazdă și pluginuri, determină arborele de execuție, controlează app domain-urile. Acest Plugin Manager determină dacă o metodă se poate executa, cum se va executa și ce se întâmplă dacă aceasta nu se execută corect. Tot acest modul creează controalele.

Language Processor este componenta care “traduce” limbajul de tip 3 într-un limbaj înțeles de framework. În esență este un analizor lexical care scoate din fraza care descrie o metodă, un șir de tokeni care apoi vor fi utilizați de PluginManager pentru a determina elementele oferite/necesare ale unei metode.

Plugin Base conține doar 2 atribute cu care trebuiesc decorate clasele. PluginClassAttribute cu rolul de a eficientiza căutarea metodelor expuse de plugin (nu se caută decât în clase publice marcate cu acest atribut) și PluginMethodAttribute cu care se marchează metodele expuse de plugin. Acesta din urmă ia un singur parametru: fraza care descrie metoda.

Pluginurile reprezintă elementele “din exterior”. Ele sunt cele care extind aplicația gazdă.

Plugin Manager

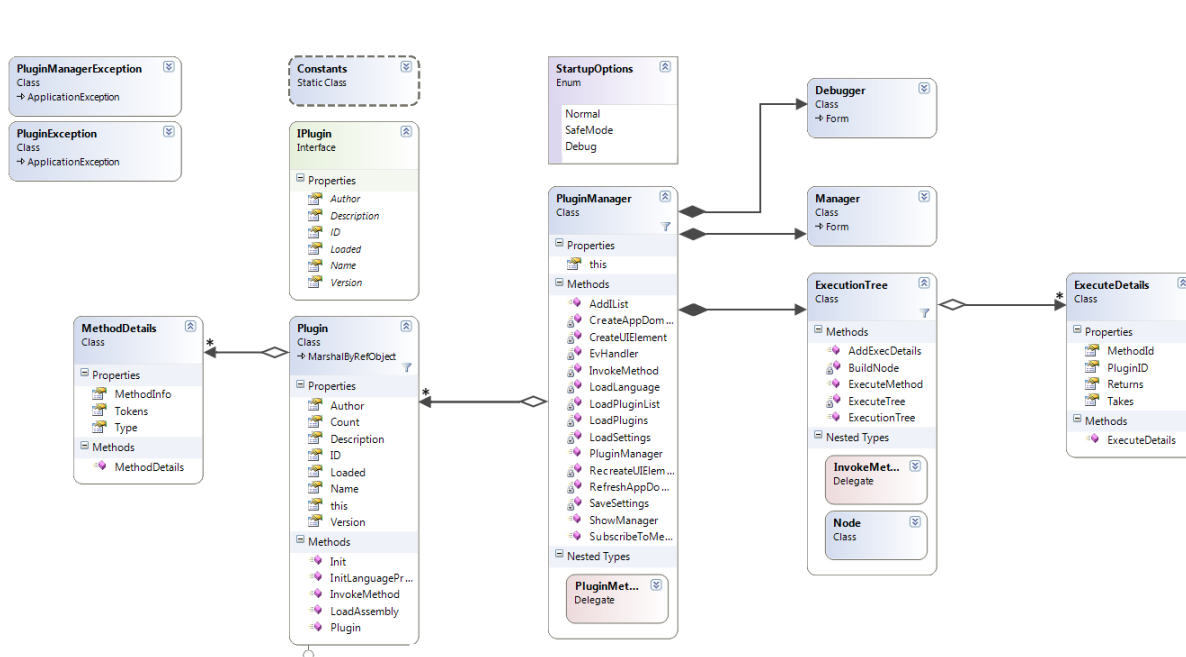


Diagrama claselor din Plugin Manager

Această componentă caută pluginuri pe disc, le verifică dacă sunt valide, le încarcă, oferă unelte pt administrarea lor, le execută și totodată comunică și cu aplicația gazdă. Totuși aceste operațiuni nu sunt executate doar de Plugin Manager singur. Încărcarea unui plugin presupune existența unor atribute (**PluginBase**) și a unui interpretor lexical (**LanguageProcessor**) deoarece se dorește analiza unei fraze scrise într-o limba necunoscută de C# sau alte limbaje de programare.

Plugin-urile

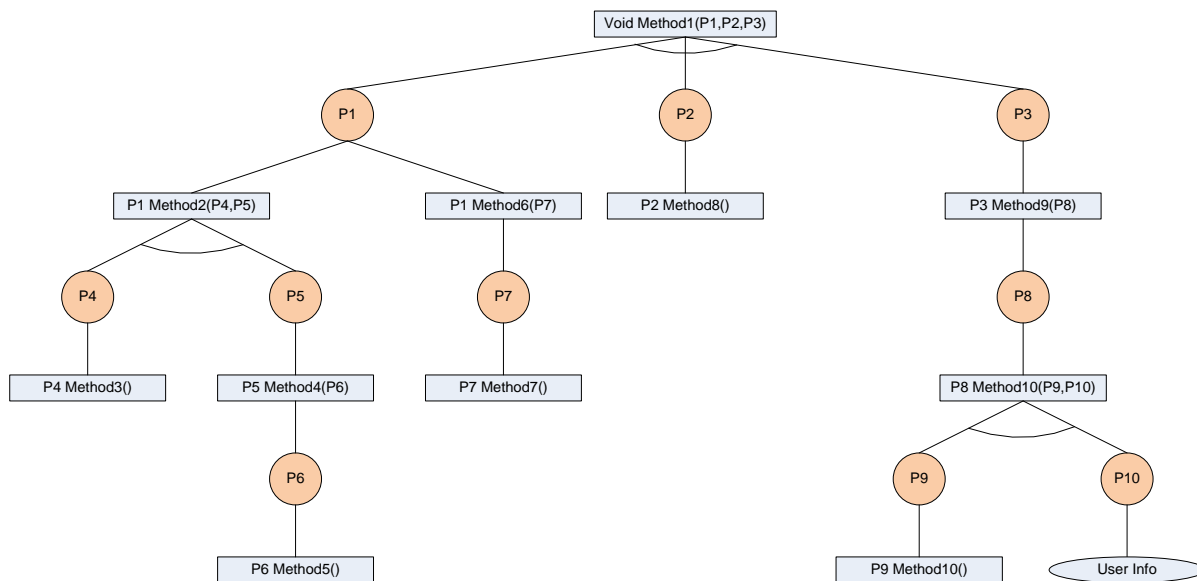
Pluginuri reprezintă extensile care se pot adăuga aplicațiilor ce folosesc Smart Extension Framework. Ele nu au o aplicație țintă ci pot fi folosite de orice aplicație care dorește acest lucru.

Plugin-urile pot fi scrise în orice limbaj care este suportat de .NET Framework: C#, VB, C++/CLI sau chiar de limbaje care nu sunt suportate de .NET prin crearea unui wrapper în unul din limbajele menționate anterior.

Ele nu au nevoie de abstract factory! Majoritatea framework-urilor pentru pluginuri folosesc desingn pattern-ul numit abstract factory care presupune (printre altele) existența unei interfețe care se fie implementată de orice plugin. Tot ce este necesar este un atribut care ia ca unic parametru un string - o frază într-un limbaj regulat.

Smart Extension Framework folosește un al doilea application domain pentru a încărca pluginurile. Acest lucru nu este un moft ci o necesitate deoarece un assembly nu poate fi descărcat din memorie decât o dată cu distrugerea app domain-ului care îl conține.

Execuția unei metode dintr-un plugin



O metodă poate fi executată conform unui arbore de execuție care este un arbore and-or. Dacă în acest arbore nu se găsesc metode care se returneze un anumit parametru atunci metoda care necesită acest parametru este ignorată. Dacă acest lucru duce la invalidarea altor noduri atunci este posibil ca metoda care a fost cerută inițial să nu poată fi executată.

În cazul în care două sau mai multe metode pot returna valoarea necesară pentru un parametru atunci se alege acea metodă care va face cele mai puține apeluri până la finalizare.

Bibliografie

- **“Components, Frameworks, Patterns” – Ralph E. Johnson**
<http://st-www.cs.uiuc.edu/users/johnson/cs329/2003/framework97.pdf>
- **“Artificial Intelligence: A Modern Approach. Second Edition” - Stuart Russell, Peter Norvig [Prentice Hall]**
http://www.amazon.com/Artificial-Intelligence-Modern-Approach-Prentice/dp/0137903952/ref=pd_bbs_1?ie=UTF8&s=books&qid=1209996022&sr=8-1
- **Plug-in Manager – Bob Aman [codeproject]**
<http://www.codeproject.com/KB/cs/dynamicpluginmanager.aspx>
- **Jason Zander's WebLog**
<http://blogs.msdn.com/jasonz/archive/2004/05/31/145105.aspx>
- **Suzanne Cook's .NET CLR Notes**
<http://blogs.msdn.com/suzcook/archive/2003/05/29/57120.aspx>
- **MSDN**
<http://msdn.microsoft.com/en-us/library/default.aspx>