Prolog Server Pages Extensible Architecture
Prolog alkalmazások illesztése webes környezethez

Hunyadi Levente

2005. november 11.

Előzmények

A web korai szakaszában kis számú felhasználó kevés távoli kiszolgálón található statikus tartalmat ér el.

Előzmények

A web korai szakaszában kis számú felhasználó kevés távoli kiszolgálón található statikus tartalmat ér el.

Rövidesen nyilvánvaló igény merül fel dinamikus tartalom generálására.

Előzmények

A web korai szakaszában kis számú felhasználó kevés távoli kiszolgálón található statikus tartalmat ér el.

Rövidesen nyilvánvaló igény merül fel dinamikus tartalom generálására.

Ezt hivatott megvalósítani a Common Gateway Interface-architektúra: a kiszolgálófolyamat a válasz összeállításához külső alkalmazást indít el megfelelő környezettel, amely kimenetére előállítja a kívánt tartalmat.

Előzmények

A web korai szakaszában kis számú felhasználó kevés távoli kiszolgálón található statikus tartalmat ér el.

Rövidesen nyilvánvaló igény merül fel dinamikus tartalom generálására.

Ezt hivatott megvalósítani a Common Gateway Interface-architektúra: a kiszolgálófolyamat a válasz összeállításához külső alkalmazást indít el megfelelő környezettel, amely kimenetére előállítja a kívánt tartalmat.

Könnyen kiterjeszthető szemlélet.

A CGI architektúra hátrányai:

- nem skálázható: új folyamat indul minden kérés kiszolgálására; nagy számú kérés esetén a terhelés túlzottá válik
- nehezen fejleszthető, karbantartható: az alkalmazáslogika, a megjelenítés és vezérlés nem különül el, hanem egyetlen alkalmazás kódjában összpontosul

Előzmények

Webalkalmazás-fejlesztésre alkalmas keretrendszerek válasza:

- hosszú életű container folyamat
- elkülönülő szöveges állományokban megtervezett megjelenítés
- a megjelenítésbe komponensekként kapcsolódó üzleti logika
- a részletek fordítással kapcsolódnak egésszé, amit a keretrendszer támogat
- a gyakran előforduló feladatok (protokollkezelés, válaszalkotás) automatizálása, támogatása, a keretrendszer elrejti a vezérlést

Áttekintés

A Prolog nyelven írt alkalmazások számára mindeddig háromféle lehetőség kínálkozott webfelület biztosítására:

- futtatás önálló HTTP kiszolgálóként (pl. SWI-Prolog `http` könyvtára)
- kapcsolódás webkiszolgálóhoz CGI felületen keresztül (pl. `PiLLoW` modul)
- beágyazás imperatív nyelvű keretrendszerbe (pl. SICStus PrologBeans Java interface)

Áttekintés

A felsorolt megoldások kétféle osztályt képviselnek:

- Prolog-alapú megoldások viszonylag alacsony vezérlési támogatással, illetve összefésülődő üzleti logikával és megjelenítéssel (Model 1 architektúra)
- imperatív nyelvű megoldások magas szintű vezérlési támogatással és elkülönülő megjelenítéssel (Model 2 architektúra), amelyben azonban a Prolog nyelvű üzleti logika csak körülményesen és korlátozottan érhető el

Kitűzött cél

Miért Prologban?

- szakértő rendszerek
- intelligens következtetések
- összefüggések feltárása

A Prolog nyelv által a webalkalmazásokba további intelligencia építhető.

Kitűzött cél:
Egy átfogó Model 2 architektúrájú keretrendszer webfelület biztosítására anélkül, hogy idegen nyelvi felületek használatára lenne szükség.

Architektúra

A Prosper architektúrája kétrétegű:

- a Prolog Web Container a vezérléssel kapcsolatos feladatokat látja el
- a Prolog Server Pages a megjelenítés és az üzleti logika szétválasztását támogatja

Prolog Web Container

Feladatai:

- a HTTP protokoll sajátosságainak elrejtése,
- a kommunikáció lebonyolítása,
- a környezeti változók (GET és POST) Prolog jól ismert modulon keresztül történő elérésének biztosítása,
- a viszony fenntartása
- alkalmazásszintű változók egyidejű módosításának szabályozása (kölcsönös kizárás)
- többszálúság problémáinak elrejtése a könyvtárhasználó elől

Prolog Web Container

A megvalósítás kihívásai:

- hosszú életűség
- skálázhatóság
- terheléselosztás
- átlátszóság

Hosszú életűség és skálázhatóság

A keretrendszer a kiszolgálóhoz CGI helyett a FastCGI protokollon keresztül kapcsolódik, ennek előnyei:

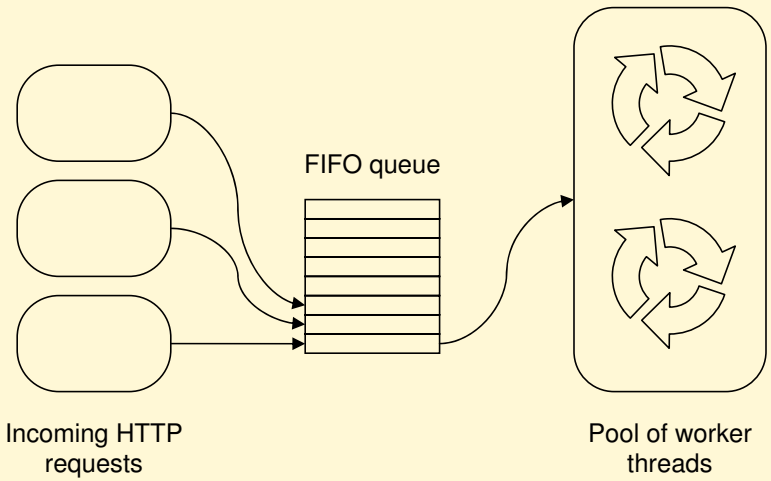
- rugalmasság: a webkiszolgáló és a keretrendszer futhat azonos vagy különböző gépeken; a kommunikáció történhet csővezetékek vagy TCP/IP kapcsolat segítségével
- skálázhatóság: a FastCGI protokoll úgy burkolja a CGI protokollt, hogy a kérések kiszolgálása között az alkalmazás nem indul újra, hanem folyamatosan fut, ezáltal lehetőség nyilik kérések közötti állapotmegőrzésére, a folyamatos újraindulási költség megtakarítására

Terheléselosztás

Az egyszerre érkező kérések kiszolgálását különböző szálak végzik. Egy hosszú időt igénylő (esetleg időkorlátot túllépő) kérés kiszolgálása így nem akasztja meg más felhasználók munkáját.

Terheléselosztás

A Prosper a *worker-thread* modellt alkalmazza a kérések elosztására: a beérkező kérést a fogadószál egy szabad feldolgozószál felé továbbítja, amennyiben van ilyen. Ellenkező esetben a kérés kiszolgálása felfüggesztődik. Ezáltal elkerülhető, hogy a már megkezdett kérések feldolgozását további kérések megakadályozzák a rendszerben életrehívott túlzott mennyiségű szállal.



Átlátszóság

A Prosper legjelentősebb előnye más megoldásokhoz képest.

- A Prolog megvalósítás elől rejtve marad minden HTTP-specifikus elem, a környezet dinamikus tényállításokon keresztül, mint *psp:http_get*/2 vagy *psp:http_post*/2 elérhető el.
- A kérések közötti hosszú életű adatok megőrzéséért a keretrendszer felel. A viszony (session) azonosítása automatikus, a változók dedikált modul predikátumain keresztül (*psp:http_session_*/n) állíthatók be, módosíthatók és törölhetők.
- Alkalmazás-hatókörű változók esetén a keretrendszer biztosítja a kölcsönös kizárást.

Prolog Server Pages

Strukturált sablontechnológiát biztosít, amely lehetővé teszi HTML, XHTML vagy XML nyelvű oldalak dinamikus létrehozását.
A sablontechnológia céljai:

- a megjelenítés és az üzleti logika teljes szétválasztása
- a Prolog nyelvhez illeszkedő deklaratív sablontechnológia
- tiszta nyelvi szerkezet
- rugalmasság, lehetőség a bővíthetőségre, kiterjeszthetőségre
- könnyű elsajátíthatóság

Sablonok

A sablontechnológia jellemzői:

- megjelenítés leírása XML formátumban
- más hasonló technológiákban is fellelhető szabványos elemkészlet a *control flow* biztosítására
  - feltétel (`psp:if`),
  - többes elágazás (`psp:choose`),
  - iteráció (`psp:for-each`, `psp:for-all`)
  - elemek dinamikus létrehozása (`psp:element`), stb.

Sablonok

- globális változók a HTTP környezet elérésére
- lokális változók számított adatok tárolására és elérésére
- típusos kifejezésnyelv egyszerűbb számítások, szövegműveletek elvégzésére, változók értékének behelyettesítésére
- változtatható kimeneti formátum (XML, XHTML vagy HTML)

Feldolgozás

A Prolog Server Pages lapok feldolgozása két fázisban történik:

- előfeldolgozás (preprocessing): a lapok beolvasása és értelmezése egyszeri költség, a Prosper a lapokat előfeldolgozott formában a memóriában tárolja
- kiértékelés (evaluation): az előfeldolgozott lapokat az adott kontextusba helyettesítve értékeli ki a Prosper, ekkor jön létre a kimenet

Ez a megközelítés váltja ki a más technológiákban alkalmazott fordítást.

Prolog Server Pages *Extensible* Architecture

Kiterjeszthetőség

A szabványos elemkészlet további elemekkel bővíthető.

- az új elemek *<névtér>*:*<elemnév>* alakban hivatkozhatók
- névterek a Prosper konfigurációs állományában regisztrálhatók
- az új elemekre ugyanúgy érvényes a kétfázisú feldolgozás:
  - *<elemnév>*(Attributes, Content, Terms) kampóeljárás az előfeldolgozáshoz, itt történhet a szintaktikai ellenőrzés, egyszerűsítő átalakítás
  - *<elemnév>*(Attributes, Variables, Content, Terms) kampóeljárás a kiértékeléshez

Üzleti logika

A megjelenítést leíró állományokhoz üzleti logika kapcsolható.

- logika megvalósítása hagyományos Prolog modulban történhet, már meglévő könyvtárakat nem kell módosítani
- a megjelenítési réteg képes listák, összetett kifejezések, rekordok (név=érték párok listája) kezelésére

Továbbfejlesztési lehetőségek

Adattartalmak rugalmas kezeléséhez a Prologot adatbázissal szükséges kiegészíteni, a jelen megoldásban ez ODBC-szinten bonyolítható le.

Továbbfejlesztési lehetőségek

Adattartalmak rugalmas kezeléséhez a Prologot adatbázissal szükséges kiegészíteni, a jelen megoldásban ez ODBC-szinten bonyolítható le.

Az ODBC-réteg a Prolog környezethez rosszul illeszkedik, használata körülményes.

Továbbfejlesztési lehetőségek

Adattartalmak rugalmas kezeléséhez a Prologot adatbázissal szükséges kiegészíteni, a jelen megoldásban ez ODBC-szinten bonyolítható le.

Az ODBC-réteg a Prolog környezethez rosszul illeszkedik, használata körülményes.

Továbbfejlesztési irány: alacsonyszintű adatbázis-elérési réteg fölé magasszintű, a Prolog nyelvbe épülő lekérdezési felület biztosítása.

Összefoglalás

A Prosper egy keretrendszer, amely

- aktívan támogatja a HTTP kérés-válasz folyamatot,
- korábbi megoldásoktól eltérően a Prolog programozó számára teljesen átlátszó,
- tiszta szerkezetű,
- szétválasztja az üzleti logikát és a megjelenítést,
- mindeközben hatékony,
- így kiválóan alkalmazható közepes méretű intelligens webalkalmazások fejlesztésében.

Elérhetőség

Referencia-implementáció, példaalkalmazás és -adatbázis:

http://sourceforge.net/projects/prospear