LearningJourney - introducere in Zope 3

Release 1.0

December 14, 2008

Contents:

Chapter 1

Prezentare generala

Zope 3 este o librarie completa ce face posibila dezvoltarea de servere de aplicatii folosind Python. O buna parte din aceste librarii pot fi folosite in orice aplicatie Python, independent de ceea ce reprezinta in mod clasic o aplicatie Zope.

Dezvoltarea Zope 3 a inceput in 2001-2002, ca o reactie la experienta castigata un urma dezvoltarii si folosirii serverului de aplicatii Zope 2. Prima versiune publica (Zope 3.0) a fost publicata in 2004.

Designul infrastructurii pe care o reprezinta Zope 3 a fost in mare parte centrat pe nevoile companiilor mari si ai clientilor Corporatiei Zope, in mod special avandu-se in vedere realizarea de aplicatii cu un nivel de logica complicata. Ca urmare, el poate fi folosit direct pentru dezvoltarea de aplicatii complexe, eficienta maxima a acestuia gasindu-se in aceste medii.

Principalele componente care fac posibile dezvoltarea de aplicatii web cu Zope sunt:

- publicarea unui graph de obiecte prin diverse protocoale, in mod tipic http
- ofera o infrastructura similara celei MVC pentru publicarea paginilor
- are un sistem de securitate ce poate fi aplicat obiectelor si paginilor
- integreaza Zope Component Architecture (ZCA)

In mod tipic o aplicatie Zope include:

- integrare cu zc.buildout pentru construirea aplicatiei
- fisiere de configurare a serverului, eventual si integrare cu PasteScript
- un modul pentru pornirea aplicatiei (startup.py)
- pachete de cod si ZCML ce implementeaza aplicatia dorita

Chapter 2

Filozofia ecosistemului Zope 3

Cand Zope 2 a fost publicat, era un produs extrem de inovativ (in perioada in care a aparut Zope, CGI-ul inca era considerat tehnologie de varf). Totusi unele dintre alegerile si presupunerile facute in design-ul Zope 2-ului nu mai sunt valabile in prezent, astfel incat Zope 3 a fost conceput in mare parte si ca reactie la neajunsurile Zope 2.

Astfel, in Zope 3 primeaza urmatoarele idei:

- este “pythonic”. Urmeaza conventiile de structurare a codului, foloseste librariile Python, pastreaza o structura simpla, usor de inteles a codului.
- se promoveaza interoperabilitatea - se ofera sisteme de adaptare a librariilor externe, este implementat standardul WSGI, se ofera flexibilitate in alegerea mediului de persistenta, etc.
- se doreste disparitia “programatorului Zope” si inlocuirea acestuia cu “programatorul de aplicatii web Python”
- codul se vrea a fi cat mai curat, elegant si testat, obiectele persistente nu sunt incarcate cu functionalitate care nu le apartine in mod logic

Un testament al flexibilitatii librariilor Zope 3 este integrarea acestora in diverse aplicatii si framework-uri, printre care Plone, repoze.bfg, vudo, etc. De altfel, cele mai multe pachete de pe Cheeseshop (PyPI) apartin comunitatilor Zope (Plone, Zope3, Grok, zc.buildout)

Chapter 3

Puncte tari, puncte slabe

Pentru a intelege cat mai bine Zope 3, este benefic sa se cunoasca avantajele si dezavantajele lui. Acestea sunt:

3.1 Puncte tari

3.1.1 Face posibila realizarea de aplicatii complexe

Unul dintre atuurile Zope 3 este ca ofera o modalitate de administrare a complexitatii. Una dintre solutiile gasite de industrie pentru realizarea de aplicatii complexe si extensibile este aceea de a baza aplicatiile pe o arhitectura bazata pe componente. Exemple includ XPCOM (Mozilla), Java Beans (Sun), COM (Microsoft) si Gumbo Flex (Adobe). Solutia oferita de catre Zope 3 este integrarea librariilor `zope.interface` si `zope.component`, adica **Zope Component Architecture - ZCA**.

3.1.2 Folosirea ZCA-ului inseamna flexibilitate (plugability)

Cea mai mare parte a librariilor Zope 3 folosesc ZCA, astfel se asigura o flexibilitate foarte mare in modificarea comportamentului librariilor: puncte de injectie si de modificare a functionalitatii acopera intreaga gama de operatiuni a Zope-ului.

3.1.3 Ofera solutii noi si inovatoare pentru dezvoltarea de aplicatii web

La fel cum Zope 2 a fost un produs inovator la vremea lui si a reusit sa se faca sustinut si folosit de mai bine de 10 ani, inovatiile aduse de catre Zope 3 il fac sa ofere o solutie puternic argumentata pentru dezvoltarea de aplicatii web. Zope 3 ofera solutii integrate pentru majoritatea problemelor tipice cu care se intalneste dezvoltatorul de aplicatii web

3.1.4 Compatibilitate WSGI

La fel ca Pylons, Turbogears si altele, Zope 3 ofera posibilitatea de a fi integrat cu librarii WSGI “out of the box”. Prin integrarea cu PasteScript se ofera o posibilitate foarte rapida de integrare cu filtre si alte “middleware” WSGI.

3.1.5 Adaptabilitate la noile tendinte

Una dintre tendintele curente este dezvoltarea de proiecte relativ mici, formate din echipe putin numeroase. In acest context sunt preferate solutiile "agile" si conventiile in defavoare explicitatii, astfel incat sa se asigure o productivitate cat mai mare cu cat mai putin cod scris. Pentru acoperirea acestui segment exista **Grok**, un framework care ofera un nivel de conventie peste explicitatea preferata de Zope 3. Pentru cei care prefera minimalismul exista **repoze.bfg**, un framework similar cu Pylons dar care integreaza librariile Zope.

3.1.6 Dezvoltatori cu experienta

Unii dintre dezvoltatorii Zope lucreaza de aproximativ 12 ani cu componentele Zope si precursoarele acestora (Bobo, Principia, etc). Unul dintre avantajele oferite de tenacitatea si persistenta in comunitate a acestora oameni este experienta pe care acestia o ofera in dezvoltarea de noi versiuni a librarii si concepte, care sa ofere solutii pentru problemele sau dificultatile intalnite in dezvoltarea de aplicatii pe platforma Zope.

3.1.7 Librariile Zope sunt refolosibile

Cunostintele dobandite in programarea pe platforma Zope pot fi refolosite in dezvoltarea de aplicatii Python: avantajele arhitecturii pe componente pot fi aplicate oricarei aplicatii Python.

3.2 Puncte slabe

3.2.1 Comunitatea de dezvoltatori/utilizatori

Desi comunitatea de dezvoltatori Zope 3 este una relativ mare si in puternica expansiune datorita adoptarii de catre Plone, in general Zope se bucura de o expunere si un marketing mai slab decat alte framework-uri web, gen Django sau Ruby on Rails.

3.2.2 Necesita dezvoltatori cu experienta, este relativ greu de invatat

Zope 3 nu e PHP. Prin folosirea unor concepte avansate si oferta de solutii pentru multitudinea de probleme des intalnite in dezvoltarea de aplicatii web, Zope 3 este relativ dificil de folosit de catre programatorii incepatori. Cu siguranta, unui incepator i-ar fi mult mai usor sa inteleaga PHP decat sa foloseasca in mod eficient infrastructura Zope 3. Un programator care nu a mai avut de-a face cu tehnologiile Zope trebuie sa invete limbajul de templating, arhitectura sistemului, modul de functionare al ZCA, dialectul ZCML XML, concepte precum viewlet-uri, librariile de formulare, etc. Cu toate acestea, pentru a fi productiv nu este nevoie de o cunoastere aprofundata a tuturor componentelor, ci doar acelea care se doresc a fi folosite.

3.2.3 Folosirea cu predominanta a ZCA

Desi nu reprezinta concepte foarte straine sau dificil de invatat pentru un programator cunoscator al sistemului OOP, ZCA necesita totusi un timp de acomodare pentru a putea intelege modul in care este trebuie folosita in avantajul programatorului.

3.2.4 Probleme cu documentatia

Aceasta este o problema de perceptie si marketing. Zope 3 nu are in acest moment un site atractiv care sa atraga si sa centralizeze o documentatie exhaustiva, asa cum are, de exemplu, Django. Cu toate acestea, exista carti pentru Zope 3 la curent cu noile modificari. Fiecare dintre pachetele ce constituie Zope 3 este bine documentat la nivel de cod, iar prin folosirea pachetului `zope.interface` se asigura o buna documentare la nivel de cod a librariilor. Astfel, API-ul Zope-ului este foarte bine documentat si poate fi accesat direct din aplicatia Zope 3 sau pe site-ul http://apidoc.zope.org. Fisierele de documentare a librariilor Zope 3 sunt adunate intr-o carte disponibila in acelasi loc.

3.2.5 Integrarea cu baze de date relationale

Desi Zope 3 ofera solutii puternice pentru integrarea cu baze de date relationale (prin zope.alchemy si STORM), faptul ca acestea nu vin integrate direct cu Zope-ul si acesta ofera in mod implicit integrarea cu ZODB poate fi perceput drept o slabiciune a Zope-ului.

Chapter 4

Facilitati incorporate

4.1 Publicarea prin web

Principalul rol al Zope-ului este acela de a publica pagini prin web. Desi nu este neaparat legat de protocolul http, o mare parte din librariile Zope se preocupa de aceasta problema. Modul in care Zope realizeaza aceasta publicare este acela de a asocia pagini unui graph (arbore) de obiecte. In principiu aceste obiecte sunt obiecte persistente provenind din ZODB, dar la fel de bine se pot publica si obiecte tranziente.

4.2 Persistenta obiectelor

In mod implicit datele sunt salvate intr-o baza de date cu obiecte, ZODB. Odata cu aparitia librariilor cu facilitati ORM au aparut si modalitati de integrare cu acestea. Una dintre problemele care o rezolva Zope-ul in integrarea cu baza de date este mecanismul de tranzactie. Daca o eroare a aparut in codul ce genereaza pagina sau proceseaza datele utilizatorului, datele din baza de date nu sunt salvate.

4.3 API pentru dezvoltarea de aplicatii

Zope 3 se bucura de un ecosistem de librarii care acopera o mare parte din ceea ce este necesar pentru dezvoltarea de aplicatii web. API-ul Zope-ului este bine documentat si este stabil. In cazul in care unele librarii se depreciaza, acestea sunt mentinute si se pastreaza compatibilitate in versiunile urmatoare. Datorita folosirii pachetelui zope.interface in marea majoritate a librariilor Zope, se asigura autodocumentarea API-ului, iar prin folosirea extensiva a modelului de programare de tip Aspect Oriented Programming, extinderea sau modificarea comportamentului implicit al librariilor este usor de realizat, la fel si dezvoltarea de cod ce va fi usor de extins de catre alte parti.

4.4 Sistem de templating extensibil

Desi Zope poate folosi orice motor de templating, in mod clasic este preferata sintaxa ZPT. In Zope exista doua implementari ZPT: cea clasica si z3c.pt, o noua librarie, compatibila cu sintaxa ZPT si integrata cu Zope, dar care ofera o viteza de cel putin 10 ori mai mare. In Zope sistemul de templating ZPT poate fi extins cu noi tipuri de expresii.

4.5 Modele de templating

Zope ofera cateva modele de nivel inalt de integrare a continutului cu template-urile: **content providerele**, ce insereaza continut in template in functie de tipul contextului, **viewlet-urile**, o forma extensibila de management al content providerelor si **pagelet-urile**, care separa continutul unui fragment html (pagina sau view) de layout-ul aceastuia.

4.6 Sistem de indexare si cautare

Zope ofera un sistem integrat de indexare a obiectelor, oferind in mod implicit indecsi pentru Text si valori arbitrare (Field). Scrierea unui nou tip de index nu este o sarcina foarte complicata, in mod special datorita API-ului simplu, ceea ce face foarte usoara integrarea cu sisteme de cautare specializate precum Lucene sau Xapian.

4.7 Adnotarea obiectelor

Unul dintre conceptele inovatoare integrate in Zope 3 este cel al adnotarii obiectelor. Acest mecanism, facilitat de ZCA, permite adnotarea cu date a oricarui obiect (fie el persistent sau tranzient), astfel incat sa poata fi extinsa functionalitatea acestora.

4.8 Internationalizare

Zope 3 ofera un mecanism de internationalizare compatibil cu standardul **gettext**. Acest mecanism are extensii pentru ZPT si se ofera si un API foarte usor de folosit din Python. De asemenea se ofera mecanisme pentru localizare (formatarea numerelor, a datelor si a timpului) precum si o baza de date ce contine diferentele regionale ce implica mecanismul de localizare. Exista de asemenea si unelte pentru dezvoltatori ce faciliteaza extragerea mesajelor internationalizabile si integrarea in fisiere de traducere deja existente. Este, de asemenea, posibila internationalizarea continutului prin folosirea a cateva extensii create de comunitatea Zope.

4.9 Evenimente

O modalitate foarte eleganta de asigurare a unor puncte de insertie pentru extinderea functionalitatii, evenimentele sunt prezente si in Zope 3. Astfel, exista o gama larga de evenimente definite si declansate de catre Zope in timpul functionarii. Printre cele mai importante sunt cele legate de ciclul de viata al obiectelor (crearea, adaugarea, modificarea sau stergerea obiectelor).

4.10 Formulare web

Zope ofera doua librarii pentru generarea, validarea si procesarea automata a formularelor web, iar comunitatea Zope a dezvoltat un nou pachet (z3c.form) care reprezinta o solutie similara celei integrate in Zope, dar mai avansata si mai flexibila.

4.11 Securitate

Securitatea este unul dintre capitolele in care Zope este foarte puternic. Sistemul integrat in Zope abstractizeaza interactiunea cu agentii externi (utilizatori) si ofera un mecanism bazat pe permisiuni, grupuri si roluri. In mod implicit exista doua nivele de verificari ale securitatii: per pagina publicata si per obiect.

4.12 Librarii de testare a codului

Zope ofera posibilitati de testare automata a codului (de altfel testarea este puternic incurajata de catre comunitatea Zope). Se ofera integrare cu toate mecanismele de test oferite de Python, putandu-se realiza atat teste pe cod cat si teste de integrare si functionalitate a intregului sistem.

4.13 Alte protocoale

Zope ofera un sistem foarte simplu pentru definirea de facilitati de trimitere asincrona a mesajelor de email. De asemenea, se ofera posibilitatea publicarii obiectelor prin protocoalele WebDAV, FTP si XMLRPC.

Chapter 5

Diferente fata de Zope 2

Programatorii familiarizati cu dezvoltarea pe platforma Zope 2 trebuie sa aiba in vedere urmatoarele:

5.1 Modificarea radicala a conceptelor folosite

Desi unele concepte din Zope si CMF se pastreaza, exista suficient de multe modificari incat sa se considere cele doua sisteme ca fiind total diferite. Cu toate acestea, un programator Zope 2 are nevoie de o perioada mai mica de invatare si acomodare (in special datorita mentinerii ZODB ca solutie pentru persistenta obiectelor)

5.2 Dezvoltarea prin web

A fost eliminata complet. Una dintre modificarile aduse de Zope 3 este refuzul dezvoltatorilor de a mai stoca cod sau template-uri in baza de date. Cu toate acestea, exista posibilitatea de a implementa forme de dezvoltare prin web folosind librariile existente in Zope.

5.3 Achizitia

Achizitia este limitata si este explicita (de altfel, nici nu este folosita de catre comunitate). Unele dintre problemele pentru care achizitia era necesara nu mai sunt valabile (dezvoltarea prin web) sau sunt rezolvate prin folosirea de alte paradigme: utilitare locale, inregistrate la nivelul site-urilor locale si un sistem inteligent de layere si skinuri implementat la nivelu codului Python.

5.4 RestrictedPython

Odata cu eliminarea dezvoltarii prin web, RPython nu a mai fost necesar. Astfel, in template-urile ZPT este permis accesul nerestrictionat la obiecte (accesul este totusi restrictionat de catre mecanismul standard al Zope-ului, ce ingradeste accesul la obiecte pe baza setarile explicite de securitate care au fost atribuite acelui obiect).

Chapter 6

O aplicatie exemplu

LearningJourney este o aplicatie minimala realizata pentru a exemplifica dezvoltarea unei aplicatii de gen CMS cu Zope 3. In principiu este foarte similara unui sistem de blogging multi-user. Ca facilitati, ofera:

- auto-inregistrarea utilizatorilor, cu un sistem de verificare a emailului
- o zona de continut personala, protejata de roluri si permisiuni, in care utilizatorul poate sa creeze continut personal
- un tip simplu de continut ce contine un camp de text ce poate fi editat cu ajutorul editorului *TinyMCE* si un camp de taguri, pentru clasificarea inregistrarii
- cautarea prin site folosind catalogul in textul inregistrarilor realizate de utilizatori
- o pagina in care se face o clasificare simpla a inregistrarilor, pe baza tagurilor

6.1 Generarea skeletului aplicatiei

Skeletul aplicatiei a fost generat folosind ZopeSkel, un proiect ce extinde PasteScript cu un template (zope_app) pentru generarea unei aplicatii Zope 3. Aplicatia generata este o aplicatie WSGI ce poate fi configurata folosind unul dintre cele doua fisiere `ini` din radacina (in functie de modul in care se doreste rularea aplicatiei - in mod *debug* sau pentru productie).

Pentru pornire se foloseste:

```
# bin/paster server debug.ini
```

sau

```
# bin/learningjourney-ctl fg
```

Ambele comenzi vor porni aplicatia in modul debug.

6.2 Structura aplicatiei

Structura generata a aplicatiei este cea a unui pachet python dezvoltat folosind setuptools. In radacina se afla modulul `setup.py` ce configureaza pachetul, iar codul in sine sta in namespace-ul `learningjourney` din directorul `src`. In namespace-ul `learningjourney` se gasesc 5 pachete, dintre care cele mai importante sunt `app` si `ui`. Primul, `app`, gazduieste modelele si logica aplicatiei,

independente in principiu de publicarea acesteia ca si pagina web. Pachetul `ui` contine tot ceea ce tine de paginile web publicate de aplicatie (pagini, imagini si resurse CSS). Fisierul `startup.py` este cel care defineste serverul de aplicatii si este folosit de catre scripturile din folderul `bin` pentru pornire.

6.3 Fisiere ZCML

In radacina aplicatiei stau cateva fisiere zcml: `site.zcml`, `apidoc.zcml` si `custom-security.zcml`. Fisierele zcml sunt fisiere XML ce contin instructiuni de configurare a componentelor folosite in codul aplicatiei si a librariilor Zope. Aceste operatiuni de configurare au direct echivalent in cod Python si pot fi realizate si prin cod, insa separarea in XML are mai multe roluri:

- permite administratorilor site-ului sa configureze aplicatia fara a necesita cunostinte de programare (se presupune ca XML-ul este un limbaj mai facil administratorilor)
- sistemul de configurare ZCML nu permite suprascrieri ale unor componente prin redefinirea lor. Pentru redefinere trebuie folosit un mecanism explicit de “override”
- in general, face mai usoara programatorului identificarea mai usoara a componentelor din cadrul unui pachet, fara a fi necesara citirea codului

Un exemplu de baza al unui fisier zcml:

```
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:browser="http://namespaces.zope.org/browser"
           package="learningjourney"
           i18n_domain="learningjourney">

   <include file="configure.zcml" />
   <include package=".ui" />
   <includeOverrides file="overrides.zcml" />

   <configure package="z3c.widget.tiny">
       <browser:resourceDirectory path="lib/resources" />
   </configure>

</configure>
```

Primul tag, `configure`, serveste drept directiva de grupare si specificare a pachetului pentru care sunt adresate directivele din interior. El defineste namespaceul XML implicit drept `http://namespaces.zope.org/zope` si namespace-ul cu prefixul **browser** drept `http://namespaces.zope.org/browser`. Acestea sunt doar cateva din namespace-urile XML folosite, insa si cele mai importante.

Directiva `include` foloseste la includerea unui nou pachet (in care este cautat un fisier cu numele configure.zcml si inclus in sistemul de configurare) sau un fisier anume, specificat cu calea completa, relativa la directorul fisierului curent. In cazul in care se foloseste un nume de pachet, acesta poate fi indicat sub forma unui de pachet python indicat absolut (numele complet al pachetului) sau relativ (se poate folosi un punct la inceput pentru identificarea unui pachet aflat in pachetul curent, sau doua puncte pentru determinarea unui pachet aflat in pachetul parinte).

Directiva `includeOverrides` serveste la identificarea unui fisier a carui configurare va avea intaietate in cazul in care se afla in conflict cu alte directive de configurare.

O referinta completa a dialectului ZCML se afla pe site-ul **zope.org** la http://apidoc.zope.org/++apidoc++/. Alte directive ZCML vor fi discutate pe parcurs.

6.4 Bazele Zope Component Architecture

Interfetele sunt unul dintre mecanismele fundamentale ale Zope 3. Ele sunt obiecte care specifica (documenteaza) comportamentul altor obiecte care declara ca le "asigura". O interfata specifica comportamentul prin:

- documentatie informala, sub forma de doc-stringuri
- definire de atribute si metode
- invariante, care sunt conditii pe care obiectele trebuie sa le indeplineasca pentru a "indeplini" acea interfata.

Astfel, o interfata descrie caracteristicile unui obiect, capacitatea lui. Ea descrie ceea ce face obiectul, nu si cum o face. Pentru a observa modul in care obiectul realizeaza ceea ce interfata declara trebuie citita implementarea obiectului, adica clasa acestuia.

Folosirea interfetelor se poate spune ca apartine unui model de programare (patern). Una dintre recomandarile facute in *biblia paternurilor* - **Design Patterns** este "sa se programeze nu pentru o implementare, ci pentru o interfata". Definirea unei interfete ajuta in intelegerea sistemelor implicate si este primul pas pentru folosirea ZCA.

Un exemplu de interfata:

```
from zope.interface import Interface, Attribute, implements, provides

class IBoardgame(Interface):
    """Un element de invatare personala  """

    title = Attribute("title")
    description = Attribute("description")

    def borrow(to_person):
        """Permite modificarea proprietarului jocului prin imprumutare """
```

Se observa:

- interfetele sunt clase ce mostenesc `zope.interface.Interface` (direct sau prin intermediari)
- atributele sunt definite ca obiecte de tip `Attribute`
- metodele nu trebuiesc implementate, ci doar definite si documentate. De aceea nu este nevoie de declararea parametrului self in semnatura metodei

O clasa ce va implementa interfata poate arata astfel:

```
class Boardgame(object):
    implements(IBoardgame)

    title = u""
```

```
    description = u""

    def borrow(self, to_person):
        self._location = "In acest moment cartea se afla la " + to_person
```

Se remarca declaratia `implements(IBoardgame)`, ce inregistreaza clasa `Boardgame` ca implementand interfata `IBoardgame`. La fel ca multe alte lucuri in Python, aceasta declaratia are la baza un contract de tip "intelegere intre gentlemeni". Ea nu obliga la implementarea cu adevarat a interfetei de catre programator, deci nu poate fi folosit la realizarea unui sistem de tip "static typing" in Python. Ea poate fi folosita, in schimb, la determinarea capabilitatilor unui obiect:

```
>>> game = Boardgame()
>>> IBoardgame.providedBy(game)
True
>>> IBoardgame.implementedBy(Boardgame)
True
>>> list(providedBy(game))
[IBoardgame]
```

De retinut: o clasa implementeaza (va avea `implements()` in cod), un obiect "asigura" (`provides()`).

`zope.interface` poate fi folosita si la decorarea cu interfete si a unor obiecte si clase care provin din librarii externe:

```
>>> classImplements(Boardgame, IBoardgame)
```

si echivalentul in zcml:

```
<class class=".app.Boardgame">
    <implements=".interfaces.IBoardgame" />
</class>
```

sau direct a obiectelor:

```
>>> alsoProvides(game, IBoardgame)
```

Odata ce se cunosc interfetele, definitia componentei in acceptiunea Zope este simpla: o componenta este un obiect care asigura cel putin o interfata. Majoritatea claselor din cadrul Zope-ului sunt scrise astfel incat ele sa devina componente atunci cand sunt instantiate.

Invariantele reprezinta o metoda de verificare a obiectului fata de interfata.

6.5 Alte facilitati oferite de ZCA

`zope.component` este o librarie bazata pe `zope.interface` care introduce cateva tipuri de componente, in fapt o implementare a unor modele de dezvoltare (design patterns). Cele 4 componente de baza sunt:

- adaptori
- utilitare

- subscribere
- handlere

6.5.1 Adapterii

Adaptorul este o implementare a modelului AOP (aspect oriented programming). El ajuta, cu ajutorul interfetelor, sa se obtina un aspect al unui obiect. In procesul de adaptere sunt implicate obiectul adaptat si interfata ce determina aspectul care ne intereseaza asupra obiectului.

Un exemplu: Sa presupunem ca avem un container in care se afla diverse obiecte. Unele sunt fisiere audio (mp3), altele sunt imagini, altele sunt fisiere text, etc. Ne intereseaza sa afisam dimensiunea specifica fiecaruia dintre aceste obiecte. Astfel, pentru fisierele audio va fi afisata marimea in timp, pentru imagini marimea in pixeli, etc.

Solutiile care nu ar implica `zope.component` pot fi:

- Construirea unei clase care sa cunoasca modul in care se extrage informatia din fiecare tip de obiect. Acest sistem este inflexibil: pentru a adauga un nou tip de obiect in acel folder, clasa ar trebui completata in asa fel incat sa stie despre acel tip.
- Implementarea, de catre fiecare obiect, a unei metode speciale care sa intoarca informatia care va fi afisata. Aceasta presupune ca acele obiecte sa stie deja in ce sisteme vor fi integrate, ce poate avea ca si consecinta supraincarcarea cu functionalitate a obiectelor.

Solutia oferita de adapteri este una eleganta, insa mai complexa. Pentru fiecare tip de obiect exista o componenta care adapteaza obiectul si extrage informatia de pe acesta.

Vom avea o interfata `IDisplaySize` care defineste modul in care se afiseaza dimensiunea obiectelor:

```
class IDisplaySize(Interface):
    """Asigura informatii despre marimea obiectelor"""

    get_size():
        """Afiseaza marimea, pentru utilizatori"""
```

In codul care realizeaza afisarea marimii, aceasta va fi extrasa de pe fiecare obiect construind un adapter pentru fiecare obiect:

```
>>> size = getAdapter(IDisplaySize, obj).get_size()
```

Se remarca ca adaptorul ce este construit in urma apelarii `getAdapter` este o componenta ce asigura (`provides`) interfata `IDisplaySize`. Prin constructia adaptorului se poate obtine o implementare specifica fiecarui tip de context:

```
class Mp3DisplaySize(object):
    zope.component.adapts(IMp3File)
    zope.interface.implements(IDisplaySize)

    def __init__(self, context):
        self.context = context

    def get_size(self):
        sound_length = extract_track_size(context)
        return "%s seconds" % sound_length
```

```
class ImageDisplaySize(object)
    zope.component.adapts(IImage)
    zope.interface.implements(IDisplaySize)

    def __init__(self, context):
        self.context = context

    def get_size(self):
        width, height = get_image_size(context)
        return "%s x %s px" % (width, height)
```

Astfel, in functie de tipul contextului (`IMp3File` sau `IImage`), va fi selectata clasa care va fi folosita in constructia adapterului, obtinandu-se astfel un obiect diferit ce va sti cum sa extraga informatia dorita din context.

Constructia `getAdapter(...)` poate prescurtata cu:

```
>>> IDisplaySize(obj).get_size()
```

Exista si multiadapteri, care adapteaza mai mult de un obiect la o anumita interfata. Cel mai des intalnit exemplu de `multiadapter` este pagina (sau view-ul), ce adapteaza request-ul - informatia provenita de la utilizator, impreuna cu obiectul context, la un o informatie de tip `IBrowserPublisher` ce va fi intoarsa utilizatorului. Paginile sunt inregistrate ca multiadapteri cu nume, asa ca nu mai trebuie sa specificam intefata la care vrem sa adaptam cele doua obiecte, pentru ca exista doar una singura pentru acel nume si acel tip de request:

```
>>> view = getMultiAdapter((context, request), name='index.html')
>>> page_content = view()
```

Restul de componente (utilitare, subcribere, handlere) vor fi discutate pe parcurs.

6.6 Publicarea unei pagini simple

Zope 3 ofera o varietate mare de modele de realizare si publicare a paginilor web. Cea mai simpla pagina nu are nevoie de cod Python, ci doar de un fisier template ZPT. Paginile sunt implementate folosind ZCA si de aceea trebuiesc inregistrate:

```
<browser:page
    name="about.html"
    for="*"
    template="pt/about.pt"
    permission="zope.View"
    />
```

Se observa numele paginii ce va fi folosit pentru publicarea paginii prin web, calea catre template-ul care va fi folosit pentru generarea paginii, permisiunea pe care utilizatorul trebuie sa o aiba pentru a o accesa precum si atributul `for`, care determina tipul contextului pentru care aceasta pagina va fi disponibila.

In cazul in care sunt declarate pagini cu acelasi nume se foloseste un mecanism de discriminare prin care este selectata clasa sau template-ul care vor fi folosite pentru generarea paginii. In discriminare vor fi interogate interfata (sau clasa) contextului precum si layer-ul (skin-ul) pe care este inregistrata pagina.

Prin folosirea mecanismului de mosternire pot fi inregistrate pagini generice care pot fi reimplementate, la nevoie, pentru tipuri de obiecte mai concrete.

Cand este vorba de context, acesta poate fi o cale cu puncte (dotted name) catre o clasa, catre o interfata sau asterixul, ce semnifica "toate tipurile de context" (sau `zope.interface.Interface`).

6.7 Publicarea resurselor

Resursele (imaginile, fisierele CSS si JS) sunt inregistrate cu ajutorul a doua taguri zcml aflate in namespaceul **browser**: `browser:resource`, ce inregistreaza o singura resursa (fisier text sau imagine) si `browser:resourceDirectory`, ce poate fi folosit pentru inregistrarea a unui director de resurse. Pentru inregistrarea unor resurse internationalizabile se foloseste tagul `i18n-resource`.

Calea catre aceste resurse este calculata ca fiind relativa la site-ul local (detalii despre site-uri mai jos), sub forma `http://localhost/mysite/@@/styles.css`

Pentru calcularea acestei cai, in template, este folosita o constructie de genul: `<script tal:attributes="src context/++resource++myscripts/jquery.js" />`

Exemple se pot gasi in fisierele `src/learningjourney/ui/configure.zcml` si `src/learningjourney/widget/addremove/configure.zcml` si fisierele template `src/learningjourney/ui/site/pt/layout.pt` si `src/learningjourney/widget/addremove/widget.pt`

6.8 Obiecte persistente

Pentru stocarea datelor in baza de date ZODB se folosesc clase care mostenesc clasa `Persistent`. Acestea sunt foarte simple, asa cum se poate observa luand ca exemplu `learningjourney.app.userhome.LearningEntry`: se mosteneste `Persistent`, se implementeaza interfata `ILearningEntry` si se implenteaza metodele si atributele definite in interfata.

Urmatoarea cerinta pentru integrarea cu Zope-ul este aceea de a face declaratii de securitate pentru obiectele generate. In cazul in care acestea nu exista, accesul la attributele obiectului va fi interzis (si va declansa errori de tip `Forbidden`). Aceasta declaratie se face in zcml:

```
<class class=".userhome.LearningEntry">
    <require
        permission="zope.View"
        interface=".interfaces.ILearningEntry" />
    <require
        permission="zope.ManageContent"
        set_schema=".interfaces.ILearningEntry" />
</class>
```

Pentru accesul direct la obiect, neintermediat de securitate, se poate folosi urmatorul exemplu:

```
>>> from zope.proxy import getProxiedObject
>> obj = getProxiedObject(someobj)
```

In cadrul tag-ului require se foloseste attributul `interface` pentru a desemna o interfata ce defineste metodele accesibile si atributele disponibile pentru citire, precum si permisiunea necesara. Se foloseste *set_schema* pentru a desemna permisiunea care este necesara pentru a modifica atributele desemnate de interfata indicata.

In cazul containerelor sunt implicate doua interfete necesare pentru desemnarea permisiunilor: IReadContainer si IWriteContainer. Exemplu:

```
<class class=".site.Application">
  <require permission="lj.ModifyContent"
    interface="zope.app.container.interfaces.IWriteContainer" />
  <require permission="zope.View"
    interface="zope.app.container.interfaces.IReadContainer" />
  <allow attributes="getSiteManager" />
  <require permission="zope.ManageServices" attributes="setSiteManager" />
</class>
```

Pentru restrictionarea tipurilor de obiecte ce pot fi adaugate intr-un container sau restrictionarea tipurilor de container in care poate fi adaugat un obiect se folosesc constrangerile din zope.container.constraints. Un exemplu se afla in *learningjourney.app.userhome*.

Alte elemente care mai pot aparea in aceasta declaratie sunt:

- tag-ul <allow>, ce poate fi folosi pentru a marca accesul la anumite atribute desemnate de o interfata sau direct prin numirea directa drept publice (necesita permisiunea zope.Public)
- optiunile `attributes`, `set_attributes` si `like_schema` care permit configurarea permisiunilor la nivel de atribute, precum si copierea setarilor de securitate ale unei alte clase.
- tag-ul `<implements`, descris mai sus, ce permite declararea implementarii unei interfete de catre o clasa, independent de codul Python

6.9 Interactiunea cu utilizatorul prin obiectul request

Mai sus am vazut cum se poate realiza o pagina simpla folosind un template. In mod normal este nevoie de un nivel de logica si interactiune mai mare cu utilizatorul. Astfel, putem folosi o clasa pentru generarea paginii, pe care o inregistram astfel:

```
class HelloWorldPage(object):

    def __init__(self, context, request):
        self.context = context
        self.request = request

    def __call__(self)
        return u"Hello world"


<browser:page
    name="hello.html"
    class=".pages.HelloWorldPage"
    for="*"
    permission="zope.View"
    />
```

Se observa ca pagina publicata este obtinuta prin apelarea instantei paginii - se apeleaza metoda `__call__` si aceasta apare ca un multiadapter pentru context si request. In realitate clasa definita aici este folosita ca *mixin* pentru generarea la *runtime* a unui nou tip, astfel incat nu este necesara mostenirea clasei `BrowserPage` sau scrierea explicita a metodei `__init__`.

Un exemplu de utilizare a unei clase in combinatie cu un template ZPT este pagina `explore.html` (`ExplorePage`), asociata obiectelor de tip `learningjourney.app.interfaces.ILearningJourneyApplication`, ce este definita in modulul `learningjourney.ui.site.page`.

In cadrul template-urilor asociate unei clase, instanta clasei si atributele ei pot fi accesate prin intermediul obiectului `view`. Contextul paginii se afla in variabila `context`, iar requestul in variabila `request`.

Deoarece obiectul **request** din cadrul paginii ofera acces la datele asociate metodei GET sau POST care a fost folosita pentru a accesa pagina, se pot implementa pagini formulare simple, folosind o metoda de genul:

```
class SampleForm(object):
    def __call__(self):
        if 'submit' in self.request.form:
            return u'Hello, %s' % self.request.form.get('name', 'John Doe')
        else:
            return ViewPageTemplateFile('sampleform.pt')()
```

Template-ul *sampleform.pt* ar contine:

```
<form method="POST">
    <input type="text" name="name" />
    <input type="submit" name="submit" />
</form>
```

Prin obiectul **request.response** se pot stabili diversi parametri ai raspunsului intors catre vizitator (de exemplu, se pot seta headere sau se poate implementa un redirect). Un exemplu simplu de astfel de formular se gaseste in clasa `EntryDeletePage` din `learningjourney.ui.homefolder.page` sau clasa `DashboardRedirect` din acelasi modul.

O conventie folosita in Zope 3 pentru a indica mecanismului de traversare ca are de-a face cu o pagina sau un view este aceea de a prefixa numele paginii in cadrul url-ului cu doua semne `@`. Impreuna formeaza doi "ochi", cu o trimitere directa la ceea ce reprezinta, un "view".

Exemplu: `http://localhost/@@index.html`

Prin folosirea acestei conventii se scurcircuiteaza mecanismul de traversare, astfel ca se incearca direct construirea unui view cu numele `index.html`, in loc sa se caute un obiect cu numele `index.html` in cadrul obiectului context curent.

De asemenea, in cadrul template-urilor, paginile pot fi inserate direct, prin apelarea de genul: `<div tal:content="structure item/@@detail" />`, unde item este un obiect persistent (ar putea fi context sau un obiect din cadrul unui repeat, iar `detail` este numele unui view inregistrat si disponibil pentru acel tip de obiect. View-ul (sau pagina) `detail` va fi construita cu obiectul `item` ca si context.

6.10 Augumentarea interfetelor cu date despre tipul atributelor

O alta librarie pilon in cadrul Zope 3-ului este `zope.schema`, o extensie a librariei `zope.interface` care permite specificarea mai exacta a tipului atributelor din cadrul interfetelor. In acest fel interfetele se transforma in "schema" si pot fi folosite pentru diverse sarcini: validarea valorilor obiectelor, generarea de formulare complexe, automate, generare de pagini de vizualizare a obiectelor, etc. Prin folosirea extensiei zope.schema, atributele devin "field"-uri, iar impreuna cu `zope.interface.fieldproperty.FieldProperty`, se poate implementa un mecanism

foarte simplu de validare automata a valorilor obiectului, pe baza schemei (vezi modul de implementare a clasei `LearningEntry` in `learningjourney.app.userhome`.

`zope.schema` defineste o multitudine de tipuri de campuri, de exemplu o linie de text (`TextLine`), o data (`Date`), o valoare de tip adevarat/false (`Bool`), o lista (`List`), etc. Bineinteles, se pot scrie si alte tipuri de campuri.

Un exemplu de interfata/schema este `ILearningEntry`, din modulul `learningjourney.app.interfaces`. Aceasta interfata este folosita pentru implementarea automata a doua formulare:

- un formulare de creare si adaugare a obiectelor de tip `LearningEntry`
- un formular de editare a obiectelor de tip `ILearningEntry`

Ambele de gasesc in modulul `learningjourney.ui.homefolder.page`

6.11 Site-uri si registre globale

In cazul in care configurarea ZCA se face prin zcml, aceasta este inregistrata de catre Zope in cadrul unui registru global. Exista totusi posibilitatea de a creea un registru ZCA "local", care sa existe la nivelul unui container persistent din ZODB. Acest container va purta numele de "site" si la nivelul lui se pot configura si inregistra setari care sunt "locale" acelui site, care vor avea precedenta fata de cele inregistrate la nivel global sau mai sus in ierarhia ZODB. Un exemplu de inregistrare a unui site local este codul aflat in `learningjourney.app.event.configure_site`, unde sunt create si inregistrate si utilitarele locale. De asemenea, caile generate pentru accesarea resurselor (CSS, imagini, JS) sunt relative la locatia acestui site.

6.12 Utilitare

Un alt tip de componente definit de `zope.component` sunt utilitarele. Acestea sunt de doua tipuri: globale si locale. In principiu, acestea se comporta similar cu obiectele de tip **singleton** din alte limbaje si modele de programare. Pe baza unei interfete se poate obtine obiectul unic ce este inregistrat ca utilitar.

Pentru inregistrarea unui utilitar global se foloseste tagul `<utility>`, in care se desemneaza interfata pe care acesta o va asigura, numele pe care il va avea si componenta care va fi folosita pentru construirea utilitarului.

```
<utility
    provides="ILanguageNegociation"
    component=".app.LanguageNegociation"
    permission="zope.Public" />
```

Pentru inregistrarea locala, la nivelul site manager-ului local, se foloseste functia `zope.component.registerUtility` sau functionalitatea din interfata ZMI aflata in pagina "Registration".

Exemple de folosire se gasesc in `learningjourney.ui.search.page.SearchPage`.

6.13 Indexarea si cautarea obiectelor

Deoarece cautarea prin obiecte dupa anumite atribute este o operatie relativ costisitoare din punct de vedere al timpului, solutia implementata in mod clasic de catre zope este cea a indexarii valorilor intr-un catalog.

Catalogul este un obiect persistent inregistrat ca utilitar local pentru interfata `zope.app.catalog.interfaces.ICatalog`, fara nume. Astfel, el poate fi obtinut folosind

```
>>> catalog = zope.component.getUtility(ICatalog)
```

In cadrul catalogului sunt adaugati indecsii (exemplu in `learningjourney.app.event`), pentru care se configureaza interfata (aspectul) obiectului pe care il indexeaza precum si atributul pe care il indexeaza. Acesti indecsi stocheaza ca referinte catre obiectul indexat un **intid** - un numar intreg, unic in cadrul bazei de date, desemnat fiecarui obiect de catre utilitarul `zope.app.intid.interfaces.IIntIds`.

Indexarea obiectelor se face atunci cand acestea sunt adaugate sau sterse din containere (pe baza evenimentelor `ObjectAddedEvent`, `ObjectRemovedEvent`), precum si atunci cand se declanseaza un eveniment `ObjectModifiedEvent`. Exemplu:

```
>>> obj.title = u"My title"
>>> zope.event.notify(ObjectModifiedEvent(obj))
```

La operatiunea de indexare a unui obiect participa fiecare index din catalog. Se ia interfata desemnata indexului si se verifica daca obiectul implementeaza acea interfata sau daca exista un adapter care sa asigure interfata pentru obiectul indexat. Odata gasit adapterul, se indexeza valoarea atributului inregistrat in configurarea indexului. Un exemplu de folosire a cautarii se gaseste in `learningjourney.ui.search.page`.

6.14 Sistemul de layere si skinuri

Librariile CMF din Zope 2 ofera conceptul de

- **layere** pe care sunt inregistrate resursele, paginile
- **skin-uri**, care grupeaza layer-ele pentru a oferi o anumita infatisare si capabilitati ale unui website.

Aceste concepte se regasesc in Zope 3 intr-o forma simplificata si usor de implementat. Un layer este determinat de o interfata ce mosteneste `zope.publisher.browser.interfaces.IBrowserRequest`. Odata definit layer-ul, obiectele de tip browser (gen pagini, view-uri si resurse) pot fi inregistrate pe acel layer folosind atributul `layer` in zcml. Un exemplu de layer se gaseste in modul `__init__` din *learningjourney.ui*

Un skin este un layer care a fost desemnat ca avand tipul `IBrowserSkinType`, plus numele care il va avea acel skin. Exemplu se gaseste in `learningjourney/ui/configure.zcml`.

Pentru selectarea skinului folosit se poate folosi un namespace traverser special in URL, de genul:

```
http://localhost:8080/++skin++lj/Application/@@index.html
```

In cadrul aplicatiei **LearningJourney** a fost folosit pachetul de extensie `z3c.layer.minimal`, care ofera un layer minimal, care nu are inregistrat decat paginile de erori. Acesta aduce ca avantaj faptul ca se vor cunoaste si controla exact paginile care sunt publicate prin web.

6.15 Folosirea macro-urilor in template-uri

Datorita faptului ca Zope 3 nu ofera publicarea directa a template-urilor si atasarea lor de orice tip de context, folosirea macro-urilor in Zope 3 este un pic mai anevoioasa.

Metoda clasica in Zope 3 de folosire a macro-urilor este inregistrarea unor pagini care ofera macro-uri intr-o lista dintr-o pagina denumita `standard_macros`. Folosirea lor poate fi observata in cadrul pachetului `learningjourney.ui.site`.

Pentru a face mai usoara utilizarea acestora, comunitatea Zope a realizat un pachet de extensie, `z3c.macro`, ce inregistreaza un nou tip de expresie **TAL**, *macro* si care poate fi folosit pentru inregistrarea directa paginilor care ofera macro-uri.

6.16 Internationalizarea interfetei

La nivelul template-urilor internationalizarea este cea "clasica", prezenta si in Zope 2, folosind atributele din namespace-ul i18n. De asemenea, in cadrul fisierelor ZCML se poate specifica domeniul in cadrul caruia vor fi traduse atributele de tipul "title" sau "label" are tagurilor din acel fisier.

La nivelul codului python se foloseste un `MessageIdFactory`, ce construieste un nou id pentru mesaje. Ca exemplu, se poate observa definirea unui messageid in modulul `learningjourney.i18n` si folosirea acestuia in `learningjourney.app.interfaces`. Pentru modificarea limbii pe baza caruia ii este prezentata interfata site-ului catre utilizator se poate folosi namespace traverserul `++lang++`, de genul:

```
http://localhost:8080/++skin++lj/++lang++ro/Application/@@index.html
```

O alta modalitate este cea de a modifica limba preferata a utilizatorului in cadrul requestului, folosind interfata `IBrowserPreferredLanguages`, putand tine astfel cont de setarile dintr-o sesiune sau un cookie.

6.17 Securitate, permisiuni

Sistemul de securitate al Zope-ului este unul relativ complex, dar este totodata flexibil si puternic. La baza sistemului de securitate stau interactiunile si principalii (obiecte reprezentand utilizatorii). Obiectele de tip **principal** sunt reconstruite pentru fiecare request, de obicei pe baza informatiei extrase din request. Utilitarul global care genereaza obiectele **principal** este inregistrat pentru interfata `zope.app.security.interfaces.IAuthentication`. Implementarea oferita de Zope se numeste **Pluggable Authentication Utility** (PAU), se gaseste in pachetul `zope.app.authentication` si ofera un sistem de pluginuri prin care se poate stabili modul in care se extrage informatia despre credentiile de autentificare ale utilizatorului si sursele de utilizatori cu care aceste informatii sunt folosite pentru a verifica veridicitatea lor. Un exemplu complet de construire a unui sistem de autentificare cu PAU se gaseste in `learningjourney.app.event`. Obiectul de tip principal este stocat in request si poate fi accesat de acolo (de exemplu, pentru afisarea in interfata a username-ului sau verificarea faptului ca utilizatorul este autentificat). Modul in care poate fi folosit se poate vedea concret in pachetul *learningjourney.ui.authentication*

Exista cateva directive ZCML pentru definirea tipurilor de principali (autentificati, neautentificati, etc) ce se pot observa in fisierul site.zcml din radacina aplicatiei. La runtime principalii pot fi construiti si pe baza unor informatii stocate in baza de date (sursa de utilizatori). Zope ofera o astfel de sursa de utilizatori in forma unui container (`PrincipalFolder`), ce stocheaza obiecte de tipul

`IInternalPrincipal` - atentie, acestea sunt doar informatii despre principali, nu principali in sine. Sursa de utilizatori si plugin-ul de extragere a secretelor de autentificare din request sunt obiecte in containerul PAU, iar utilitarului PAU i se specifica pluginurile de tip *extragere* si cele de tip *sursa de utilizatori*, precum si si ordinea lor. Toate acestea se pot observa in `learningjourney.app.event`

Accesul la obiecte si la proprietatile acestora se protejeaza cu permisiuni, asa cum s-a putut observa in capitolul *Obiecte persistente*. Pentru usurarea managementului permisiunilor se foloseste conceptul de roluri: un principal, intr-un anumit context, poate avea un anumit rol. Acelui rol i se pot permite anumite permisiuni. Pentru alocarea unui rol unui principal (utilizator), se adapteaza contextul la interfata IPrincipalRoleManager, astfel:

```
>>> IPrincipalRoleManager(context).assignRoleToPrincipal('lj.Owner',
                                                         request.principal.id)
```

Declararea de noi permisiuni, roluri, precum si aprobarea permisiunilor pentru roluri se poate observa in fisierul `custom-security.zcml` din radacina aplicatiei.

Chapter 7

Modalitati de rezolvare a unor probleme tipice

7.1 Folosirea interfetelor de tip marker pentru definirea capabilitatilor

O interfata de tip marker este o interfata "goala, ce mosteneste direct *zope.interface.Interface*. Folosind aceste interfate pot fi definite capabilitati si implementarea abstractizata a acestora. Declararea de catre o clasa a implementarii acelei capabilitati atrage cu aceasta si beneficierea de implementarea acelei capabilitati. Exemplu:

Sa presupunem ca vrem sa dezvoltam o solutie reutilizabila de adaugare de comentarii. Fiecare comentariu va fi stocat intr-un obiect de tip `IComment`, iar aceste comentarii vor fi stocate in anotarile obiectelor. Intr-un sistem de tip CMS, pentru care tip de obiecte va fi afisat butonul *Add comment*? A lua ca indiciu faptul ca un obiect este anotabil nu este de ajuns: nu toate obiectele anotabile trebuie sa fie comentabile. Solutia este simpla: vom crea o interfata marker:

```
class IHasComments(Interface):
    """Pe acest tip de obiecte pot fi adaugate comentarii"""
```

Adapterul folosit pentru factory-ul ce genereaza anotarea obiectelor poate fi:

```
class IComments(Interface):
    """Comentariile obiectului"""

class Coments(BTreeContainer):
    implements(IComments)
    adapts(IHasComments)

    def add_comment(self, *args, **kwds):
        self[u"Comment %s" % len(self)] = Comment(*args, **kwds)
```

Daca avem un obiect de tip BlogEntry, este suficient sa il marcam ca fiind "comentabil":

```
class BlogEntry(Persistent):
    implements(IBlogEntry, IHasComments)
```

Un viewlet care prezinta un fragment cu comentariile pe obiectul curent, plus butonul de "Adauga comentarii" ar putea arata astfel:

```
class CommentsViewlet(BaseViewlet):

    @property
    def available(self):
        return IHasComments.providedBy(self.context)
```

Iar codul care adauga comentarii poate fi pur si simplu:

```
IComments(some_blog_entry).add_comment(...)
IComments(some_other_object_type).add_comment(...)
```

7.2 Adnotarea obiectelor

Un alt concept introdus de Zope 3 in scopul de a mentine obiectele cat mai "curate" este acela de adnotare. Cu ajutorul adnotarii pot fi atasate date suplimentare obiectelor. De exemplu, sa presupunem ca avem obiecte de tipul `Boardgame`:

```
class IBoardgame(Interface):
    name = zope.schema.TextLine(title=u"Numele jocului"

class Boardgame(Persistent):
    implements(IBoardgame)

    name = u""
```

Dorim sa atasam o lista cu toate datele cand acel joc a folosit. Aceste date stau in obiecte de tip `UsageInfo`:

```
class IUsage(Interface):
    date = Date(title=u"Data folosirii")

class Usage(Persistent):
    date = None

    def __repr__(self):
        return "Used at %s" % self.date
```

Lista o vom implementa annotand obiectele de tip `Boardgame` cu informatie de tip `IUsageInfo`:

```
class IUsageInfo(Interface):
    usages = List(  title=u"Dates when used",
                    value_type=Object(title=u"Usage",
                                      schema=IUsage)
                 )

class UsageInfo(Persistent):
    implements(IUsageInfo)
    adapts(IBoardgame)

    usages = None
```

Obiectele de tip UsageInfo vor fi stocate in annotarea obiectelor de tip `IBoardgame`. Adnotarea in sine este un adapter ce se construieste folosind functia `factory` din `zope.annotation.factory`

```
>>> from zope.annotation.factory import factory
>>> usage_info_annotation = factory(UsageInfo)
>>> zope.component.provideAdapter(usage_info_annotation)
```

De obicei configurarea ultimei linii se face in zcml astfel:

```
<zope:adapter factory=".annotations.usage_info_annotation" />
```

In final, folosirea adnotarii in cod este foarte simpla:

```
>>> usage_info = IUsageInfo(some_boardgame)
>>> usage_info.usages.append(Usage(datetime.datetime.now()))
>>> print usage_info.usages
```

Adnotarea nu este ceva care sa fie in mod implicit asigurat tutoror obiectelor persistente. Pentru ca adnotarea sa fie disponibila pentru un obiect, acesta trebuie sa fie adaptabil la interfata `IAnnotations` (de exemplu, ar putea sa existe un adaptor care sa stocheze adnotarea obiectelor intr-un RDB). In cazul obiectelor persistente bazate pe ZODB, acestea pot fi facute usor adaptabile prin marcarea claselor acestora ca implementand interfata `zope.annotation.interfaces.IAttributeAnnotatable`. Exista un adapter care adapteaza acest tip de obiecte la interfata IAnnotations prin stocarea adnotarilor intr-un atribut `__annotations__`.

7.3 DublinCore

Informatia de tip `DublinCore` este stocata de catre libraria `zope.dublincore` implicit pentru toate obiectele persistente adnotabile. De aceea, pentru a beneficia de `DublinCore` pentru obiecte este suficient sa se marcheze clasa unui obiect ca implementand `IAttributeAnnotatable`.

DublinCore reprezinta o sumedenie de atribute si informatii, de aceea, daca se doreste modificarea comportamentului implicit, procesul de implementare a acestei interfete trebuie optimizat. Exista doua modalitati de a realiza aceasta:

Primul mod, se poate declara un attribut ca fiind o proprietate `DublinCore`:

```
from zope.dublincore.property import DCProperty
class Book:
    implements(IBook)
    name = DCProperty("title")
    authors = DCProperty("creators")
```

Al doilea mod implica crearea manuala a adaptorului DC folosind un *factory*, caz in care putem specifica o mapare intre campurile DC si campurile obiectului.

```
dc_annotation = partialAnnotatableAdapterFactory({
                                'name':'title',
                                'author':'creators'
                                })
```

Adapterul este inregistrat astfel:

```
<zope:adapter
    for=".interfaces.IBook"
    factory=".annotations.dc_annotation"
    provides="zope.dublincore.interfaces.IZopeDublinCore"/>
```

7.4 Folosirea relatiilor intre obiecte

Avand un folder cu imagini de pe circuitele de curse de masini, dorim sa asignam o lista de imagini unor obiecte de tipul Pilot, Echipa sau Stadion:

```
class IHasImages(Interface):
   """marker, objects have pointers to images"""

class Pilot:
   implements(IHasImages, IAttributeAnnotatable)
   #e gresit sa pui IHasImage sa faca inherit la AttributeAnnotatable
   #HasImages tine de comportament, AttributeAnnotatable tine deja de
   #implementarea anotarilor

class Team(Persistent):
   implements(IHasImages, IAttributeAnnotatable)

class Stadium(Persistent):
   implements(IHasImages, IAttributeAnnotatable)

class PicturesAlbum(BTreeContainer):
   contains(IImage)

class IPicturesInfo(Interface):
   images = List(title=u"Image",
                 value_type=Relation(title="Relation")
                 )

class PicturesInfoAnotation(Persistent):
    implements(IPicturesInfo)
    adapts(IHasImages)

    images = None

from zope.annotation.factory import factory
annotation_factory = factory(PicturesInfoAnotation, 'images_pointers')
```

Astfel, codul ce foloseste aceasta anotare este simplu:

```
IPicturesInfo(pilot_instance).images.append(img)
del IPicturesInfo(team_instance).images[somename]
```

7.5 Internationalizarea continutului cu z3c.language.switch

Implementarea de mai jos este una reala, folosita intr-un portal realizat cu Zope 3. Metoda queryAttribute a interfetei II18n a fost redefinita pentru a introduce un mecanism prin care se intoarce valoarea

limbii default a atributului interogat, in loc de o valoare goala.

```
class BusinessI18NInfo(Persistent):
    """The business content object"""

    name = FieldProperty(IBusiness['name'])
    address = FieldProperty(IBusiness['address'])
    description = FieldProperty(IBusiness['description'])
    promo_message = FieldProperty(IBusiness['promo_message'])


class Business(I18n, BTreeContainer, Contained):
    """ """
    _factory = BusinessI18NInfo
    _defaultLanguage = 'en'

    implements(IBusiness)

    name = I18nFieldProperty(IBusiness['name'])
    address = I18nFieldProperty(IBusiness['address'])
    description = I18nFieldProperty(IBusiness['description'])
    promo_message = I18nFieldProperty(IBusiness['promo_message'])

    def queryAttribute(self, name, language=None, default=None):
        #override so that we never return empty stuff
        value = super(Business, self).queryAttribute(name, language, default=None)
        if value is None:
            negotiator = getUtility(INegotiator, context=self)
            lang = negotiator.serverLanguage
            try:
                value = self.getAttribute(name, language=lang)
            except KeyError:
                pass
        if value is None:
            #try to return something meaningful
            langs = self.getAvailableLanguages()
            if langs:
                lang = langs[0]
            try:
                value = self.getAttribute(name, language=lang)
            except KeyError:
                pass
        if value is not None:
            return value
        else:
            return default
```

7.6 Modificarea traversarii cu z3c.traverser

Exemplul de mai jos foloseste un caz in care o serie de imagini sunt stocate intr-un folder adnotare al obiectelor de tip `Business`. Datorita acestui amplasament al imaginilor, pentru ca acestea sa fie publicate prin web ca aflandu-se intr-un container *images* aflat in adnotarea obiectelor de tip `Business`, a fost nevoie de modificarea mecanismului de traversare al obiectelor de tip `Business`. Pachetul `z3c.traverser` introduce un mecanism prin care se pot scrie plugin-uri pentru modificarea traversarii, per obiect, prin modificarea publisher-ului default al obiectelor cu unul care are acest suport pentru

pluginuri. Prima inregistrare zcml modifica publisher-ul pentru obiectele de tip `IBusiness`, iar cea de-a doua inregistreaza pluginul cu acest publisher (acest mecanism este descris in amanunt in documentatia `z3c.traverser`).

```
<view
  for=".interfaces.IBusiness"
  type="zope.publisher.interfaces.browser.IBrowserRequest"
  provides="zope.publisher.interfaces.browser.IBrowserPublisher"
  factory="z3c.traverser.browser.PluggableBrowserTraverser"
  permission="zope.Public">
  setup plugin traversal for the IBusiness
</view>

<subscriber
  for="lovely.reviewportal.app.interfaces.IBusiness
       zope.publisher.interfaces.browser.IBrowserRequest"
  provides="z3c.traverser.interfaces.ITraverserPlugin"
  factory=".traversing.BussinesTraverserPlugin" />


class BussinesTraverserPlugin(ContainerTraverserPlugin):
    """Traversing to business/images will return the annotation of
    BusinessImagesAlbum for the Business:
    """

    def publishTraverse(self, request, name):
        if name == "images":
            images = IBusinessImagesAlbum(self.context)
            proxied_images = LocationProxy(images, container=self.context,
                                           name="images")
            return proxied_images
        if name == "promotions":
            promotions = IPromotions(self.context)
            proxied_promotions = LocationProxy(promotions,
                                               container=self.context,
                                               name=name)
            return proxied_promotions

        subob = self.context.get(name)
        if subob is None:
            raise NotFound(self.context, name, request)
        return subob
```

7.7 Metode avansate de integrare a template-urilor

In cazul dezvoltarii unui site cu Zope 2, modul in care template-urile trebuie create este evident: este necesar un template pentru layout-ul site-ului, se folosesc macro-urile si sloturile pentru a umple acest template si eventual se foloseste mecanismul CMF de suprascriere a template-urilor folosind skinurile. Plone este un exemplu de aplicatie care a folosit acest mecanism cu succes.

Cu Zope 3, avand in vedere multitudinea de alegeri ce pot fi facute, acest mecanism nu este foarte clar conturat. In continuare vom analiza cateva dintre aceste optiuni, in contextul unui scenariu ce presupune realizarea unui website pentru o companie multinationala.

7.7.1 Inserarea continutului HTML direct

Aceasta metoda de includere a continutului din alta pagina (sa-i spunem template deocamdata) este usor similara cu functia `include()` din PHP. In Zope exista totusi avantajul ca pagina respectiva este direct legata de tipul obiectului asupra caruia ii este aplicata. Exemplu:

```
<div tal:replace="structure context/@@footer" />
```

Simplu, dar cu cateva probleme: pe fiecare pagina a site-ului va trebui copiata structura de baza a site-ului si apoi inserate partile specifice ale paginii in zone clar demarcate. Pentru orice site cu mai mult de cateva pagini devine foarte dificila modificarea structurii de baza a site-ului, pentru ca trebuiesc modificate toate paginile din site.

7.7.2 METAL: Macro-uri si sloturi, la fel ca in Plone-ul clasic

Metoda clasica de separare a layout-ului site-ului de cel al continutului paginii este acela de a folosi extensia METAL, cu macro-uri si slot-uri. Un exemplu:

In primul avem template-ul site-ului, denumit template.pt

```
<html metal:define-macro="page">
    <head metal:define-slot="header">
        <title>Some title</title>
    </head>
    <body metal:define-slot="body">
        Body content comes here
    </body>
</html>
```

Apoi trebuie sa facem macro-ul disponibil. Vom realiza o pagina cu numele de `view_macros` si o vom adauga la tuple-ul `page_macros` al view-ului `standard_macros`. Acest view este un browser-view special ce implementeaza interfata `zope.interface.common.mapping.IItemMapping` (a se vedea, de exemplu, `standard_macros.py` din `zope.app.basicskin` si `zope.app.rotterdam`. Acest view are o lista de nume de pagini care contin macro-uri si o lista de aliasuri dintre macro-uri. In final, macro-ul este inclus in pagina, astfel:

```
<html metal:use-macro="context/@@standard_macros/page">
    <head metal:fill-slot="header">
        <title>MyTitle</title>
    </head>
    <body metal:fill-slot="body">
        Content here...
    </body>
</html>
```

O alta modalitate de "gasire" a macro-urilor in cadrul unui template este acela de a pune o referinta catre template-ul cu macro-uri in cadrul clasei ce genereaza pagina:

```
class MainPage(BrowserPage):
    macros = ViewPageTemplate('/path/to/macros.pt')
```

In cadrul template-ului asociat paginii MainPage, se poate introduce macro-ul cu:

```
<div metal:use-macro="view/macros/some_macro">
```

7.7.3 z3c.macro: inregistrarea simplificata a macro-urilor

Folosind z3c.macro, registrarea de noi macro-uri devine o sarcina simpla, ce nu mai presupune redefinirea si modificarea unei clase. De exemplu, pentru inregistrarea unui nou macro `page` din `template.pt`, trebuie inserat in zcml:

```
<configure xmlns:z3c="http://namespaces.zope.org/z3c">
    <z3c:macro template="template.pt" name="page" />
</configure>
```

Macro-ul este apoi inserat in pagini cu:

```
<html metal:use-macro="macro:page">
...
</html>
```

7.8 Viewlet-uri, content providers

Luand ca exemplul site-ul companiei multinationale, sa presupunem ca vom avea un menu de navigare pentru site. Daca ar fi sa il dezvoltam site-ul folosind doar macro-uri, am scrie un macro pe care l-am insera in template-ul principal. Ce s-ar intampla daca am dor sa modificam acest menu de navigare pentru cateva dintre paginile site-ului? Doua solutii exista:

- sa inseram o gramada de logica in macro, care sa verifice cazurile speciale. Urat, complicat, e de dorit sa se evite acest caz
- sa restructuram menu-ul sub forma unui view care sa poata fi redefinit per tip de context. Aceasta solutie functioneaza in cazul in care dorim sa redefinim modul in care arata menul in functie context, dar nu si daca dorim sa modificam menu-l in functie de pagina in care apare.

Solutia oferita de Zope 3 este o varianta a celei de-a doua solutii de mai sus, prin introducerea unui tip de view ce tine cont si de view-ul in care este inserat, denumite **content providere**. Gratie interfetelor, este foarte usor sa se redefineasca continutul provider-ului in functie de tipul contextului, tipul request-ului (skin-ul) si tipul paginii (interfata sau clasa implementata de pagina). De exemplu, in site-ul nostru de text am putea defini un menu de navigare ce va fi inserat pe fiecare pagina astfel:

```
from zope.contentprovider.interfaces import IContentProvider
from zope.publisher.interfaces.browser import IDefaultBrowserLayer
from zope.publisher.interfaces.browser import IBrowserView

class MainSiteNavigation(object):
    implements(IContentProvider)
    adapts(Interface, IDefaultBrowserLayer, IBrowserView)

    def __init__(self, context, request, view):
        self.context = context
        self.request = request
        self.__parent__ = view
```

```
    def update(self):
        pass

    render = ViewPageTemplateFile('navigation.pt')
```

Acest content provider este inregistrat astfel:

```
<adapter factory=".browser.MainSiteNavigation" name="main_site_navigation" />
```

Pentru suprascrierea provider-ului, de exemplu pentru obiectele de tip PressRelease, vom folosi:

```
class PressReleasesNavigation(object):
    adapts(IPressRelease, IDefaultBrowserLayer, IBrowserView)
    render = ViewPageTemplateFile('press_releases_navigation.pt')


<adapter
    factory=".browser.PressReleasesNavigation"
    name="main_site_navigation" />
```

Astfel, folosirea content provider-ilor permite "componentizarea" paginilor web, prin dezvoltarea de componente ce pot fi refolosite in mai multe locatii si tipuri de obiecte context. Folosind inregistrarea acestui multiadaptor cu un nume, el poate fi inserat in template-uri folosind expresia *provider*:

```
<div tal:content="structure provider:lj.MyProvider" />
```

Pentru a genera continutul content provider-ilor se foloseste un proces numit "two phase rendering". Prima data se apeleaza metoda *update* al CP-ului, apoi metoda *render* ce genereaza continutul ce va fi inserat in pagina.

7.8.1 Viewlet-uri si viewlet managere

Viewlet-urile reprezinta un pas inainte in directia facute de content providere: un viewlet manager este un content provider ce agregheaza si insereaza in pagina o serie de "mini-view-uri", definite ca multi-adaptori pentru context, request, view si manager. Folosind mecanismul de viewlet-uri se poate decupla continutul de template-ul sau contextul in care este inserat: se pot controla ce "box-uri" sa apara in fiecare pagina doar prin adaugarea de inregistrari de viewlet-uri, nemaifiind necesara editarea de macro-uri, templateuri sau prea mult cod.

Pachetul zope.viewlet ofera doua noi tag-uri ZCML: browser:viewletManager si browser:viewlet. Atunci cand se inregistreaza un viewlet manager poate fi specificata atat clasa care il genereaza cat si un template, astfel incat sortarea viewlet-urilor si modul in care sunt ele inserate in pagina pot fi modificate dupa necesitati.

Desi poate fi tentanta, ideea de a transforma o intreaga pagina si intreg site-ul intr-o structura bazata pe viewlet-uri este periculoasa: formularele in viewlet-uri sunt destul de greu de implementat, paginile vor fi greu de definit si greu de administrat (deoarece nu exista o imagine clara, in cod, a ceea ce apare pe pagina). E recomandat ca macar partea principala de continut a paginii sa nu fie definita in viewet-uri, ci folosind mecanismul de pagelet-uri, descris mai jos.

Un exemplu practic:

- se defineste o interfata marker pentru viewlet manager:

```
from zope.viewlet.interfaces import IViewletManager
class IExtraStuffBox(IViewletManager):
    '''Viewlets for the extra stuff box'''
```

- se inregistreaza viewlet manager-ul:

```
<browser:viewletManager
    name='zope3tutorial.ExtraStuffBox'
    provides='.interfaces.IExtraStuffBox'
    permission='zope.View'
    layer='.demoskin.IMySkin'
    />
```

- se introduce in template-ul principal al site-ului:

```
<div tal:replace="structure provider:zope3tutorial.ExtraStuffBox">
    A box for extra stuff
</div>
```

- se scrie un viewlet:

```
>>> class SizeViewlet(object):
...
...     def __init__(self, context, request, view, manager):
...         self.__parent__ = view
...         self.context = context
...
...     def update(self):
...         pass
...
...     def render(self):
...         return size.interfaces.ISized(self.context).sizeForDisplay()
...
>>> zope.component.provideAdapter(
...     SizeViewlet,
...     (IFile, IDefaultBrowserLayer,
...      zope.interface.Interface, interfaces.IViewletManager),
...     interfaces.IViewlet, name='size')
```

In zcml inregistrarea poate fi facuta astfel:

```
<browser:viewlet
    name="size"
    for="IFile"
    manager="interfaces.IViewletManager"
    class="SizeViewlet"
    permission="zope.View"
    />
```

- Viewlet-ul poate fi declarat si doar folosind un template

```
<browser:viewlet
    name="fortune"
    for="*"
    manager='.interfaces.IExtraStuffBox'
    template='fortune.pt'
```

```
    layer='.demoskin.IMySkin'
    permission='zope.View'
    />
```

7.8.2 Separarea inregistrarii templat-ului de clasa cu z3c.viewtemplate

Continuand cu implementarea studiului nostru de caz, sa presupunem ca aceasta companie are multiple website-uri, generate de catre aceeasi aplicatie Zope, cate unul pentru fiecare tara, continutul este aproximativ identic, dar se folosesc layout-uri si template-uri usor diferite. In acest caz, sistemul de skinuri multiple aplicate aplicatiei ar functiona bine, cu singura problema fiind data de necesitatea subclasarii, per skin, a fiecare clase de view, pentru a putea modifica template-ul folosit.

Una dintre solutiile posibile este data de catre pachetul z3c.viewtemplate, ce permite separarea inregistrarii template-ului folosit de inregistrarea paginii, ce permite redefinirea simpla a template-ului folosit.

Ca exemplu, sa presupunem ca dorim sa schimbam prima pagina a unuia dintre skinuri, pentru a adauga o noua coloana. Avem `MainSitePage` ca pagina principala, cu template-ul folosit `main_site.pt` si dorim sa modificam template-ul. Va trebui sa schimbam. Pentru a beneficia de `z3c.viewtemplate`, `MainSitePage` trebuie modificata in genul urmator:

```
class MainSitePage(object):
    template = RegisteredPageTemplate()

    def __call__(self):
        return self.template()
```

sau, pur si simplu, putem mosteni `BaseView` din `z3c.viewtemplate`:

```
class MainSitePage(BaseView):
    ...
```

Apoi putem inregistra template-ul, per layer:

```
<browser:template
    for=".browser.MainSitePage"
    template="main_page.pt"
    layer=".SkinLayerOne" />
```

Se pot suprascrie, de asemenea, template-urile si pentru viewlet-uri, daca se foloseste un superclass de genul:

```
class BaseViewlet(object):

    template = RegisteredPageTemplate()

    def render(self):
        return self.template()
```

In practica, structura va arata astfel:

- un template principal al site-ul ce va asigura layout-ul si va insera viewlet managerele. Acest layout va fi oferit sub forma unui macro denumit `page`
- paginile site-ului vor folosi macro-ul `page` si vor insera doar continutul specific al paginii

Folosind un mecanism inteligent de mostenire, este posibil sa se reduca la minim necesitatea definirii de template-uri noi.

7.8.3 z3c.template, o versiune imbunatatita a pachetului z3c.viewtemplate

z3c.template este un pachet similar cu z3c.viewtemplate (permite separarea codului de inregistrarea template-ului), insa are ca scop si separarea definitiei si inregistrarii layout-ului paginii de continutul si template-ul continutului acesteia.

Sa presupunem ca implementam un site folosind z3c.template si avem pagina de stiri pentru presa. Pentru fiecare pagina vom avea un template pentru layout si unul pentru continut, insa putem sari definirea template-ului de layout daca mostenim o clasa de baza. Template-ul de layout va contine:

```
<html>
<head>
    <title tal:content="view/title" />
<head>
<body>
    <div tal:replace="view/render" />
</body>
</html>
```

Acest template va fi inregistrat:

```
<configure xmlns:z3c="http://namespaces.zope.org/z3c">
    <z3c:layout template="main_template" for=".interfaces.ISitePage" />
</configure>
```

Avem nevoie de un browser view care sa stie cum sa foloseasca template-ul de layout si cel de continut:

```
class SitePage(BrowserPage):
    zope.interface.implements(ISitePage)

    template = None
    layout = None

    title = None

    def update(self):
        pass

    def render(self):
        if self.template is None:
            template = zope.component.getMultiAdapter(
                      (self, self.request), IContentTemplate)
            return template(self)
        return self.template()

    def __call__(self):
        self.update()
        if self.layout is None:
            layout = zope.component.getMultiAdapter((self, self.request),
                          interfaces.ILayoutTemplate)
            return layout(self)
        return self.layout()
```

Clasa noastra pentru stiri de presa va mosteni clasa `SitePage`:

```
class PressReleaseViewPage(SitePage):

    @property
    def title(self):
        return u"Press release: " + self.context.title
```

Apoi putem sa inregistram pur si simplu template-ul pentru zona de continut a paginii:

```
<configure xmlns:z3c="http://namespaces.zope.org/z3c">
    <z3c:template template="press_review_view.pt" for="IPressReview" />
</configure>
```

Desi mecanismul este relativ simplu si usor de inteles, presupune o oarecare munca pentru a suporta form-uri, si de ce sa scriem clasa SitePage, cand exista deja un pachet care o ofera? Acesta este...

7.8.4 z3c.pagelet

Acest pachet introduce un nou tip de browser page: pagelet-ul. Un pagelet este o pagina cu un template pentru layout: se defineste layout-ul folosind mecanismele introduse de `z3c.template`, dar clasa `SitePage` nu mai este necesara deoarece este asigura de pachet. In interiorul template-ului de layout, acolo unde urmeaza a fi inserat continutul, se insereaza pagelet-ul, ca si content provider:

```
<div tal:replace="structure provider: pagelet" />
```

Alte “bunatati” incluse in pachet sunt:

- pachetul include integrare cu clase de formulare ale `zope.formlib`
- folosind z3c.skin.pagelet se poate porni cu un skin de start care are toate clasele necesare pentru realizarea unui website bazat pe pagelet-uri (incluzand, de exemplu, pagini de exceptii implementate cu pagelet-uri)

Pentru inregistrarea unui pagelet se foloseste (se observa similitudinea cu inregistrarea unei pagini):

```
<z3c:pagelet
    name="index.html"
    for=".interfaces.PressRelease"
    class=".views.IndexPagelet"
    layer=".interfaces.ICompanyWebsiteLayer"
    permission="zope.View"
    />
```

7.8.5 Alte pachete folositoare in template-uri

- z3c.pt: o implementare a template-urilor ZPT cu mult mai rapida (x10).
- z3c.macroviewlet -Permite definirea de macro-uri din template-uri ca viewlet-uri, inlesnind astfel scrierea de site-uri complete in viewlet-uri
- z3c.formui: integreaza pagelet-urile cu libraria z3c.form

7.9 Sfaturi generale

- codul sursa al Zope-ului este simplu si usor de interpretat. Daca nu exista documentatie, citeste codul sursa, incepand cu interfetele si apoi fisierul configure.zcml, pentru a intelege ce ofera acel pachet. Foloseste instrumentele APIDOC si Zope Book pentru a citi documentatia. Foloseste pagina de introspectare a obiectelor pentru a intelege structura obiectelor.
- Foloseste namespace-urile
- Separa partea de "backend" de cea de "frontend" in sub-pachete-uri diferite: *app* si una din variantele: *browser*, *skin*, *ui*.
- Pastreaza o balanta intre necesitatea de a separa in pachete si un numar prea mare de pachete, cu interdependinte ridicate
- Foloseste pachetele comunitatii Zope (z3c), acestea sunt cel mai adesea mai noi si mai flexibile decat cele cu care vine zope. Exemplu: z3c.form, z3c.pagelet, z3c.table, etc.
- Atunci cand faci design-ul unei librarii sau componente reutilizabile, este foarte usor sa se creeze o infrastructura bazata pe ZCA, cu multe puncte de insertie. O balanta trebuie mentinuta, pentru a nu face ca folosirea acelui pachet sa impuna necesitati ridicate (gen: implementeaza un adaptor pentru interfata asta, am nevoie de un utility pentru interfata asta, etc). Se poate intampla ca atunci cand dezvoltatorul sa vrea sa foloseasca componenta respectiva sa ii fie mai usor sa o inlocuiasca cu totul, asa ca fii sigur ca asiguri cel putin un punct unic de "override", care sa faca posibila inlocuirea intregului mecanism. Cu toate acestea:
- Incearca sa faci componentele general valabile, pentru a fi reutilizabile. Pe termen lung acesta este unul dintre avantajele folosirii Zope 3: componentele sunt usor de dezvoltat astfel incat ele sa fie refolosibile
- Viewlet-urile si pageleturile sunt o idee buna, insa nu trebuiesc abuzate, pentru ca necesita mai multa munca din partea dezvoltatorului.
- Nu e nevoie de o integrare foarte mare cu ZMI.
- Nu e recomandabil designul unei aplicatii complexe fara experienta prealabila.

Chapter 8

Ecosistemul Zope 3

Marimea ecosistemului pachetelor de extensie pentru Zope 3 este o dovada clara a vitalitatii platformei. Mai mult, datorita compatibilitatii WSGI este posibila extinderea ecosistemului pentru a include librarii pentru dezvoltarea de aplicatii web care aparent nu au legatura cu Zope. Se pot identifica 3 mari actori in acest ecosistem: Zope Corporation, Zope 3 Community (o comunitate informala, raspunzatoare in mare parte pentru realizarea Zope 3) si comunitatea dezvoltatorilor pe platforma Plone, desi pachetele realizate de acestia, compatibile pe de-intregul cu Zope 3 sunt mai putine.

8.1 Pachetele Zope

Librariile cele mai folosite pentru implementarea de aplicatii Zope 3 sunt:

- zope.annotation
- zope.contentprovider
- zope.viewlet
- zope.copypastemove
- zope.event
- zope.securitypolicy
- zope.formlib
- zope.interface
- zope.schema
- zope.app.catalog
- zope.app.container

8.1.1 Pachetele din namespace-ul zope.*

zope.annotation face posibila atasarea de date pentru obiecte. Ofera o implementare ce stocheaza aceste date intr-un atribut pe obiecte

zope.browser contine cateva interfete folosite in cadrul altor pachete

zope.cachedescriptors ofera memoizarea atributelor

zope.component implementeaza ZCA. Ofera adapter, utilities, subscribers, handlers. In implementare sunt folositi registri pentru inregistrarea componentelor, oferind totodata si posibilitatea de a crea registri locali, care sa aiba intaietate fata de cei globali (prin mecanismul de site-uri)

zope.configuration defineste un sistem de configurare extensibil. Implementeaza bazele ZCML

zope.contentprovider asigura posibilitatea de a componentiza structura paginii cu fragmente dinamice de continut, ce pot fi refolosite si legate de tipul contextului, al request-ului si al paginii in care sunt inserate

zope.contenttype utilitare pentru determinarea continutului fisierelor, extinde modulul standard mimetypes

zope.copypastemove ofera suport pentru operatiuni de copiere, pastare si mutare a obiectelor. Genereaza evenimente atunci cand se executa operatiunile de mai sus

zope.datetime definitii si utilitare pentru manipularea de obiecte de timp si data

zope.deferredimport Asigura suport pentru optimizarea operatiunilor de import, ce ajuta in pornirea mai rapida a aplicatiilor

zope.deprecation defineste un API util in marcarea unor module sau functii ca fiind depreciate

zope.documenttemplate un motor de templating folosind standardul DTML

zope.dottedname permite rezolvarea definitiilor de obiecte folosind nume cu puncte

zope.dublinecore implementare generala a standardului DublinCore, ofera si o implementare ce stocheaza aceste informatii in anotare

zope.error implementeaza utilitare globale si locale de inregistrare si raportare a exceptiilor

zope.event implementeaza un sistem de notificare folosind sistemul de subscriberi cu care se definesc handler-ele pentru acele evenimente

zope.exception contine definitii pentru cateva exceptii de baza precum si utilitare pentru formatarea exceptiilor

zope.filereprezentation defineste interfete folosite pentru reprezentarea obiectelor prin protocoale de comunicare de gen WebDav sau FTP

zope.formlib o librarie generala de generare, validare si procesare automata a formularelor pentru pagini web

zope.hookable asigura posibilitatea unui monkey-patching la runtime, global, insa explicit

zope.i18n implementarea de baza a suportului pentru internationalizare in zope, implementeaza domeniile de traducere, cataloagele de mesaje, implementeaza o extensie zcml pentru inregistrarea cataloagelor de mesaje si contine utilitare pentru formatarea internationalizata a unor mesaje precum numere, date, bani, etc.

zope.i18nmessage implementeaza mesajele internationalizate

zope.index implementeaza indecsi ca si structuri de BTree-uri, optimizate pentru indexarea unor diverse tipuri de date: campuri de valori, text si liste de valori. Acesti indecsi sunt folositi in implementarea catalogului

zope.interface librarie de baza in implementarea Zope 3, implementeaza definirea, folosirea si interogarea obiectelor si a interfetelor pe care le asigura. Reprezinta o piesa de baza in implementarea ZCA

zope.lifecycleevent defineste o serie de evenimente in care poate fi implicat un obiect (crearea, stergere, modificare, adaugare intr-un container)

zope.location defineste un model de localizare fizica a obiectelor folosind numele si parintele acestora, precum si diverse utilitare pentru localizare. Defineste de asemenea utilitare pentru plasarea obiectelor intr-un graph folosind LocationProxy, simuland astfel localizarea lor fizica. LocationProxy este important pentru ca se poate folosi pentru publicarea prin web a obiectelor nepersistente si astfel se pot realiza modele de traversare si URL-uri care nu reprezinta neaparat structura fizica a bazei de date

zope.minmax Defineste un model de rezolvare a conflictelor din infrastructura MVC folosind valorile printr-o politica ce favorizeaza valorile minime sau maxime ale obiectelor implicate in conflict

zope.pagetemplate implementeaza pagini ce folosesc template-uri cu sintaxa TAL

zope.proxy o implementare optimizata in C a conceputului de proxy. Este folosit in celelalte librarii in implementarea un proxiuri de locatie si securitate

zope.publisher implementeaza mecanismele de publicare a continutului prin protocoalele http, ftp si xmlrpc, defineste componentele implicate (request, response, skinuri, internationalizare, etc) si totodata asigura implementarea suportului WSGI

zope.rdb integrare cu baze de date relationale prin realizarea si mentinerea unor conexiuni globale cu acestea

zope.schema extinde zope.interface pentru a implementa tipuri specifice de atribute (gen numar, o linie de text, o lista de valori, etc). Tot aici sunt definite si vocabularele, utilitare ce intorc o sursa de valori pentru afisarea de optiuni in interfetele cu utilizatorii. Unul dintre pachetele de baza a Zope-ului atunci cand vine vorba de implementarea de interfate cu utilizatorii.

zope.security defineste un sistem de securitate ce foloseste principali si permisiuni pentru a restrictiona accesul la obiecte si atributele acestora. Contine integrarea cu ZCML a definitiei de permisiuni si politici de securitate.

zope.securitypolicy defineste politica de securitate a zope-ului, ce extinde infrastructura de securitate cu roluri si grupuri pentru a atinge o flexibilitate mai mare. Defineste modul in care se stocheaza rolurile si permisiunile pe obiecte folosind maparile si managerii de securitate. Contine integrarea cu ZCML a definitie de roluri si de acordare de permisiuni rolurilor si principalilor.

zope.sendmail defineste utilitare globale folosite pentru sisteme directe sau optimizate pentru trimiterea de emailuri si integrarea in zcml a acestora

zope.sequencesort utilitare pentru sortarea de liste

zope.server contine implementari de servere pentru ftp si http. ZServer (aceasta implementare) reprezinta una dintre cele mai rapide variante de server http implementat in python

zope.session implementeaza sesiuni si utilitare pentru identificarea clientilor

zope.size defineste modul in care se obtine informatia despre marimea unui obiect, precum si o implementare de baza

zope.structuredtext un motor de transformare a formatului Structured Text in html

zope.tal o implementare a sintaxei de templating TAL

zope.tales contine extensii ale sintaxei tal ce introduce expresii noi pentru traversarea obiectelor

zope.testbrowser contine un browser ce poate fi programat, putand fi folosit in definirea de teste functionale

zope.testing contine utilitare ce pot fi folosite in construirea de teste pentru aplicatii si pachete Zope. Contine suport pentru teste unitare si teste de integrare.

zope.testrecorder contine un proxy web ce poate fi folosit pentru inregistrarea comunicatiei http in scopul definirii de teste de integrare

zope.thread defineste modul in care pot fi create si administrate obiecte locale per thread. O simpla extensie a modulului standard thread

zope.traversing defineste utilitare si namespace-uri pentru traversare, precum si utilitare pentru calcularea URL-ului absolut al unui obiect contextul hosting-ului virtual al aplicatiei

zope.viewlet extinde conceptul de content providere cu cel de viewleturi si viewlet managere, ce permit definirea de zone in care vor fi inserate viewlet-urile, view-uri dependente de context si zona in care vor fi inserate. Defineste, de asemenea, cateva viewlet managere standard

zope.xmlpickle pickle based serialization to and from xml

8.1.2 Pachetele din namespace-ul zope.app

zope.app.apidoc face posibila introspectarea obiectelor pentru prezentarea unei pagini autogenerate de documentatie. Ofera o extensie ZCML pentru inregistrarea documentatiei intr-o asa numita “Zope Book”.

zope.app.aplicationcontrol ofera posibilitatea controlului aplicatiei de catre utilizator (oprire/ repornire/impachetarea bazei de date, etc)

zope.app.appsetup ofera o modalitate de configurare si construire a aplicatiei Zope

zope.app.authentication ofera un sistem complex, bazat pe plugin-uri de extragere a datelor de autentificare si autentificarea utilizatorilor (inclusiv sursa de utilizatori)

zope.app.basicskin ofera un skin simplu, cu intentia de a servi ca skelet

zope.app.boston un skin complet, similar cu Rotterdam, bazat pe viewlet-uri

zope.app.broken integreaza obiectele stricate din baza de date in asa fel incat ele sa fie vizualizate si diagnosticate

zope.app.cache ofera un utilitar global in care pot fi stocate date in scopul cachingului

zope.app.catalog o solutie completa de indexare si cautare a obiectelor. Ofera doua tipuri de indexi, bazati pe zope.index: text si camp

zope.app.component Extensii ale zope.component. Integreaza diversele componente ale zope-ului (view-urile, layerele, securitatea, utilitare, etc) si defineste extensii zcml pentru acestea.

zope.app.container Ofera o implementare bazata pe BTrees a containerelor, defineste modul de traversare a acestora precum si suport pentru preconditii (ce tipuri de obiecte pot fi adaugate, etc).

zope.app.content Defineste interfata IContentType cu care pot fi marcate alte tipuri de interfete. Defineste un vocabular ce listeaza aceste tipuri de interfete.

zope.app.dav Ofera un server webdav integrat cu restul infrastructurii Zope, ofera si integrare cu ZCML

zope.app.debug ofera un debugger care poate fi pornit din scriptul controler al aplicatiei

zope.app.dependent ofera o cale de a marca dependente intre obiectele din graph

zope.app.dtmlpage ofera un content type bazat pe template-uri DTML

zope.app.exception ofera pagini pentru exceptii, generand echivalentul in exceptii http

zope.app.external o integrara minimala ce ofera infrastructura necesara integrarii unui editor extern

zope.app.file doua noi tipuri de continut: File si Image precum si integrarea acestora cu infrastructura de reprezentare externa (IFileRepresentation)

zope.app.folder ofera un content type de tip folder, integrat cu interfata de management si reprezentarea ca obiect extern (IFileRepresentation)

zope.app.form libraria "clasica" de formuri a zope-ului, este considerata depasita datorita dependentei de zcml si a inflexibilitatii. Cu toate acestea, modelul de widget-uri este folosit de catre zope.formlib si de aceea ramane o librarie importanta.

zope.app.ftp ofera un model de view pentru protocolul ftp

zope.app.generations face posibila migrarea bazei de date atunci cand schema obiectelor persistente se modifica

zope.app.homefolder ofera o implementare a unui homefolder pentru utilizatori

zope.app.http integrare a protocolului http: exceptii, metode, definirea modului de traversare si rezolvare a view-ului

zope.app.i18n contine implementarea unui message catalog persistent precum si extensii ZCML pentru inregistrarea unui director ce contine traduceri

zope.app.i18nfile contine implementarea internationalizabila a fisierelor si imaginilor stocate in baza de date

zope.app.interface ofera un vocabular pentru interfetele oferite de un obiect

zope.app.interpretor contine un interpretor pentru cod python in care nu se poate avea incredere

zope.app.intid un utilitar ce asociaza id-uri unice obiectelor din baza de date

zope.app.keyreference un adapter pentru obiectele persistente care extrage pozitia lor din baza de date pe baza caruia i se poate asocia un intid

zope.app.locales contine traduceri si utilitare pentru software-ul Zope 3

zope.app.locking o implementare pentru blocarea accesului la obiecte, de exemplu pentru integrarea cu protocolul Webdav

zope.app.module o implementare de module python persistente (cod python in ZODB)

zope.app.onlinehelp ofera o infrastructura de documentatie si help precum si o extensie ZCML ce face posibila asocierea de catre orice pachet de sectiuni de help

zope.app.pagetemplate face posibila integrarea template-urilor ZPT cu clasele de view, ofera o expresie ZCML ce face posibila inregistrarea de noi tipuri de expresii ZCML.

zope.app.preference implementeaza o modalitate de stocare a preferintelor pentru fiecare utilizator si contine integrare cu ZCML pentru inregistrarea grupurilor de preferinte

zope.app.preview ofera o modalitate simpla de a previzualiza site-ul intr-un iframe

zope.app.principalannotation ofera o modalitate de asociere de date pentru principali, prin anotarea persistenta a acestora

zope.app.publication integreaza metodele de publicare a continutului, ofera o extensie ZCML pentru dezvoltarea de noi moduri de publicare, per tip de verb http folosit.

zope.app.publisher implementeaza obiecte/pagini, ce pot fi folosite in publicarea de continut. Implementeaza extensii zcml pentru inregistrarea paginilor si a menurilor din ZMI

zope.app.pythonpage implementeaza un tip de continut ce poate interpreta continut python

zope.app.renderer infrastructura pentru transformarea unui tip de continut intr-un alt tip de continut (de exemplu din rest sau stx in html)

zope.app.rotterdam cel mai folosit skin Zope, implementeaza o interfata completa pentru managementul zope-ului

zope.app.schema integreaza zope.schema cu infrastructura de securitate, ofera un registru pentru vocabulare

zope.app.security implementeaza infrastructura de securitate a Zope-ului, ofera metode de baza pentru autentificarea utilizatorilor si generarea de principaluri, ofera extensii zcml pentru declararea de principaluri si grupuri globale

zope.app.securitypolicy ofera pagini prin care se pot modifica local, in baza de date, per obiect, setarile de securitate

zope.app.server implementeaza si face posibila configurarea de servere http si wsgi, implementeaza o infrastructura pentru controlul acestora

zope.app.session ofera suport pentru sesiuni pentru asocierea de date cu un client

zope.app.sqlscript implementeaza obiecte de continut care pot executa comenzi sql

zope.app.testing ofera utilitare pentru crearea de teste

zope.app.tree implementeaza un widget arbore ce poate fi folosit in interfetele cu utilizatorii pentru reprezentarea graph-ului de obiecte

zope.app.twisted implementeaza un server bazat pe Twisted precum si infrastructura necesara pentru controlul si configurarea acestuia

zope.app.undo implementeaza pagini pentru controlul si executarea pasilor de undo

zope.app.winservice ofera un serviciu integrat in sistemele de operare bazate pe WinNT

zope.app.workflow ofera un motod de workflow si suport pentru crearea de definitii persistente, in baza de date, a acestora

zope.app.wsgi ofera un obiect aplicatie WSGI pentru integrarea cu protocolul WSGI

zope.app.xmlrpcintrospection ofera introspectie protocolului XMLRPC, prin implementarea unor noi metode

zope.app.zapi grupeaza o serie de functii des folosit pentru convenienta dezvoltatorilor

zope.app.zcmlfiles ofera cateva fisiere zcml care grupeaza tematic pachetele zope, permitand astfel incarcarea selectiva a acestora

zope.app.zopeappgenerations migreaza baza de date de la versiuni mai vechi ale zope-ului

zope.app.zopetop un skin Zope, nu este foarte popular, fiind o versiune mai veche

zope.app.zptpage implementeaza template-uri persistente in baza de date

8.2 Alte pachete

Exista aproximativ 300 de pachete de extensie in svn.zope.org, neincluzand namespace-urile zope.* si zope.app.*. De altfel, comunitatea Zope este cea mai mare producatoare de pachete de tip egg (dupa statisticile de pe PyPi). Printre aceste pachete, se numara solutii pentru:

stocarea fisierelor de dimensiuni mari z3c.extfile, z3c.blobfile

reutilizarea view-urilor prin customizarea template-ului z3c.template

separarea continutului unui view de layout-ului lui z3c.layout, z3c.pagelet

formulare avansate, extensibilile z3c.form

memoizarea view-urilor lovely.responsecache

headere pentru caching z3c.caching z3c.responseheaders z3ext.cacheheaders

ETAG support z3c.conditionalviews

diverse field-uri si widgeturi z3c.schema.* z3c.widget.*

integrare "first class citizen" cu RDB-uri STORM zope.sqlalchemy, z3c.saconfig, ore.alchemist

includerea automata a resurselor (css, js) zc.resourcelibrary z3c.resourceinclude

concatenarea resurselor z3c.resourcecollector

configurare ZCA "locala" z3c.baseregistry

engine de templating mai rapid z3c.pt

workflow hurry.workflow

configurare negociere, internationalizare continut z3c.language.*

framework pentru operatiuni de configurare, specifice unui pachet z3c.configurator

dezvoltarea de aplicatii zc.buildout, pb.recipes.pydev (pt Eclipse)

executarea de taskuri asincron lovely.remotetask, zc.async

partajarea sesiunii intre clientii zeo lovely.session

generarea flexibila de tabele si listing-uri z3c.table, zc.table

control mai mare asupra procesului de indexare si catalogare z3c.indexer

8.3 Alte resurse pe web

Zope 3 wiki Desi pare putin atractiv, wiki-ul Zope 3 reprezinta in acest moment singurul site oficial si este un bun punct de plecare catre alte site-uri. Locatie: http://wiki.zope.org/zope3

Noul site zope.org Desi nu este inca lansat, noul site are deja suficiente pagini noi si interesante (de exemplu, ghidul Getting Started updatat la noile metode de instalare Zope 3). Locatie: http://new.zope.org/

A Comprehensive Guide to Zope Component Architecture Un document bogat in exemple si explicatii despre Zope Component Architecture, configurarea cu zcml si modul in care poate fi folosita in orice aplicatie Python, nu doar Zope. Locatie: http://www.muthukadan.net/docs/zca.html

Ghid al pachetelor 3rd party Zope Un ghid/referinta al pachetelor de extensie din repozitorul svn.zope.org. Locatie: http://wiki.zope.org/zope3/Zope3PackageGuide

What's new in Zope 3.3 Un document care discuta modificarile aduse de Zope 3.3. Util prin faptul ca discuta succint sistemul de skinuri, vocabulare si factories. Locatie: http://kpug.zwiki.org/WhatIsNewInZope33

Zope 3 Book Desi pare invechita, o buna parte din ceea ce exista in aceasta carte este inca valabil. Locatie: http://wiki.zope.org/zope3/Zope3Book

Worldcookery Sectiunea Downloads contine codul sursa sub licenta GPL, si reprezinta o foarte buna resursa pentru intelegerea Zope. Locatie: http://worldcookery.com/Downloads

Documentatia pentru z3c.form z3c.form include documentatia direct in codul sursa al pachetului. O varianta html, generata de Sphinx se afla la adresa http://www.carduner.net/docs/z3c.form/

Chapter 9

Viitorul platformei Zope 3

Zope 3 este constituit din librarii mature, intarite de anii de folosinta in diverse aplicatii online, care nu mai necesita modificari majore. Cu toate acestea, se bucura de prezenta unui grup de dezvoltatori cu experienta, implicati in proiecte care solicita dezvoltarea platformei de baza si a noi librarii conexe. Numarul de dezvoltatori pe platforma Zope este in crestere, datorita:

- migrarii spre platforma Zope 3 a unor sisteme populare precum Plone
- aparitia initiativei Repoze, cu multitudinea de librarii Zope reimpachetate si gata de a fi folosite in alte medii (precum Turbogears)
- au aparut noi proiecte de tip CMS (z3ext, vudo, hivurt, etc) care ruleaza folosind librariile Zope

9.1 Directii noi de dezvoltare

- Se urmareste micsorarea interdependentelor dintre pachetele Zope. Aceasta va avea ca efect imbunatatirea performantelor (utilizare RAM mai mica, etc) si scaderea complexitatii vizibila dezvoltatorilor.
- Promovarea de solutii alternative la cele oferite de Zope in mod clasic, gen librarii compatibile WSGI si integrarea cu RDB-uri folosind sisteme ORM
- Coborirea nivelului de intrare ale noilor dezvoltatori prin platforma Grok, ce se doreste a fi o solutie simpla si puternica de dezvoltare de aplicatii web, dar complet compatibila cu Zope 3
- Repoze.bfg, o noua platforma de dezvoltare de aplicatii web, bazata pe librariile Zope, WebOb si WSGI, cu o integrare destul de puternica a conceptelor din Zope 3, dar cu intentia de a reduce interdependenta dintre componente si de a fi cat mai agnostic in alegerile facute, oferind multiple cai pentru rezolvarea unor probleme (gen persistenta datelor, motorul de templating, etc).
- Integrarea pachetelor chameleon ce ofera o mai buna performanta pentru template-uri precum si suport pentru sintaxe alternative, gen genshi