プログラマでなくてもわかる Zope 3 の紹介

Open Source Conference 2009 Tokyo/Spring
2009年2月20日 11:15-12:00

山本 烈 (a.k.a. Retsu)

iccm inc. 代表
Plone研究会
日本Zopeユーザー会(JZUG)

はじめに

Zope 3は大きなフレームワークなので、
今日は特徴的な部分を重点的に紹介

特徴

- 再利用性
- セキュリティ
- 品質（テスト）

今日の流れ

- Zopeとは、Zope 2とは？
- Zope 3とは？
- 再利用性
- コンポーネントアーキテクチャ
- インタフェース
- アダプタ
- ZCML
- セキュリティ
- 品質（テスト）
- その他もろもろ

Zopeって何？

Zopeって何？

PythonでWebアプリケーションを構築するための基盤ソフトウェア
Zope自体はアプリケーションではない。

誰が開発している？

Zope Corporationと世界中のZope開発者たちが開発、運営

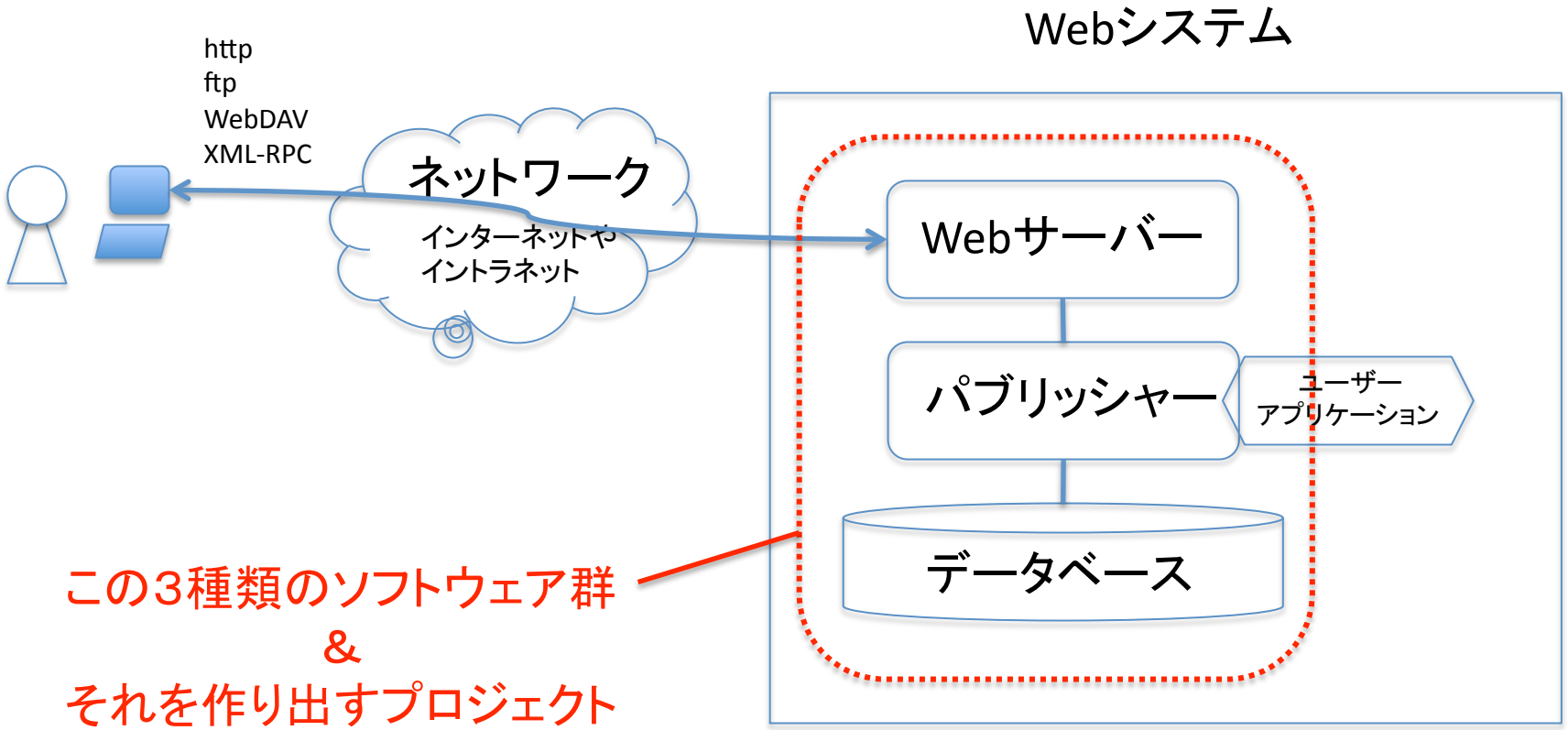
経緯は？

1998年にZope 1.9デビュー, 翌年にZope 2.0がリリース
投資家の説得によりプロプラエタリソフトをオープンソースとして公開
目的は基盤ソフトウェアの開発および維持コストの低減。
現在の最新版はZope 2.11とZope 3.4の２系列

特徴

名前どおり、オブジェクトパブリッシング
Z Object Publishing Environment
オブジェクトデータベース

Zopeとは


Webシステム
http
ftp
WebDAV
XML-RPC
ネットワーク
インターネットや
イントラネット
Webサーバー
パブリッシャー
ユーザー
アプリケーション
データベース
この3種類のソフトウェア群
&
それを作り出すプロジェクト

何が革新的？

オブジェクト中心。オブジェクトは複数の機能を同時に持てる。

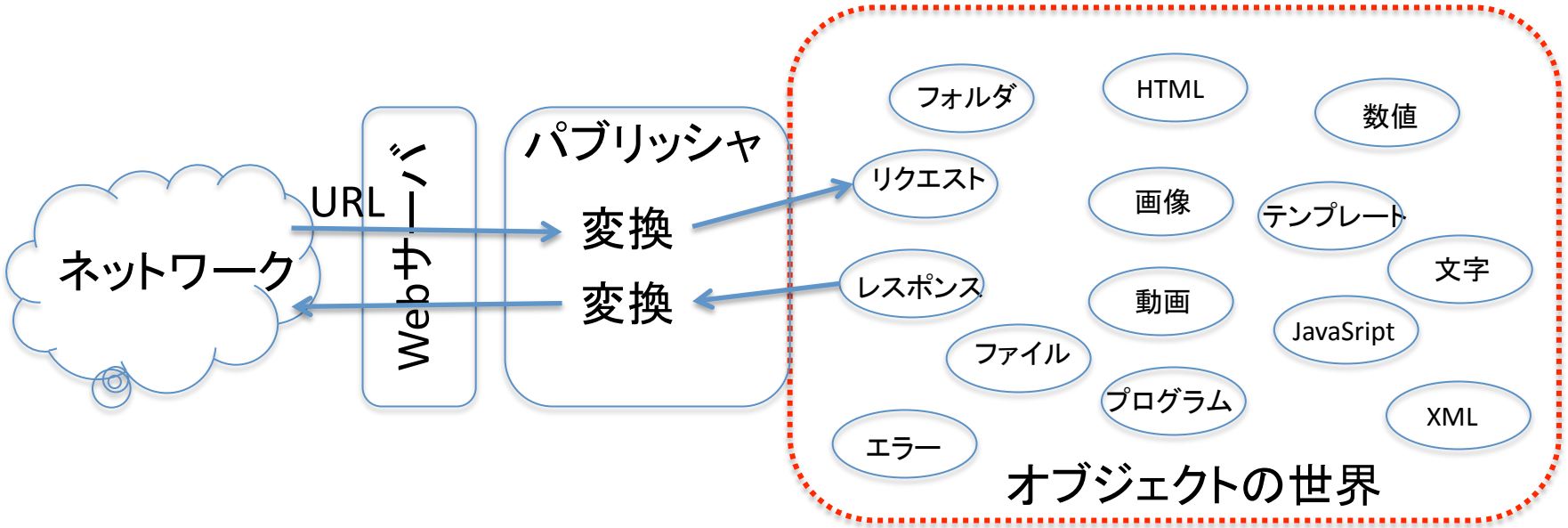
- フォルダのように何か他のものを含み、
- ファイルのように何かその中身を返し、
- プログラムのように何かを処理して返し、
- プログラムのように他のオブジェクトを呼び出せる。

トラバーサルは柔軟にオブジェクト間の関係を定義でき、
「獲得」と呼ばれるトリックも！

オブジェクトデータベース(ZODB)

- 永続的な性質のオブジェクトを自動的にディスクに格納
- アプリケーションプログラマは入出力処理を書かない

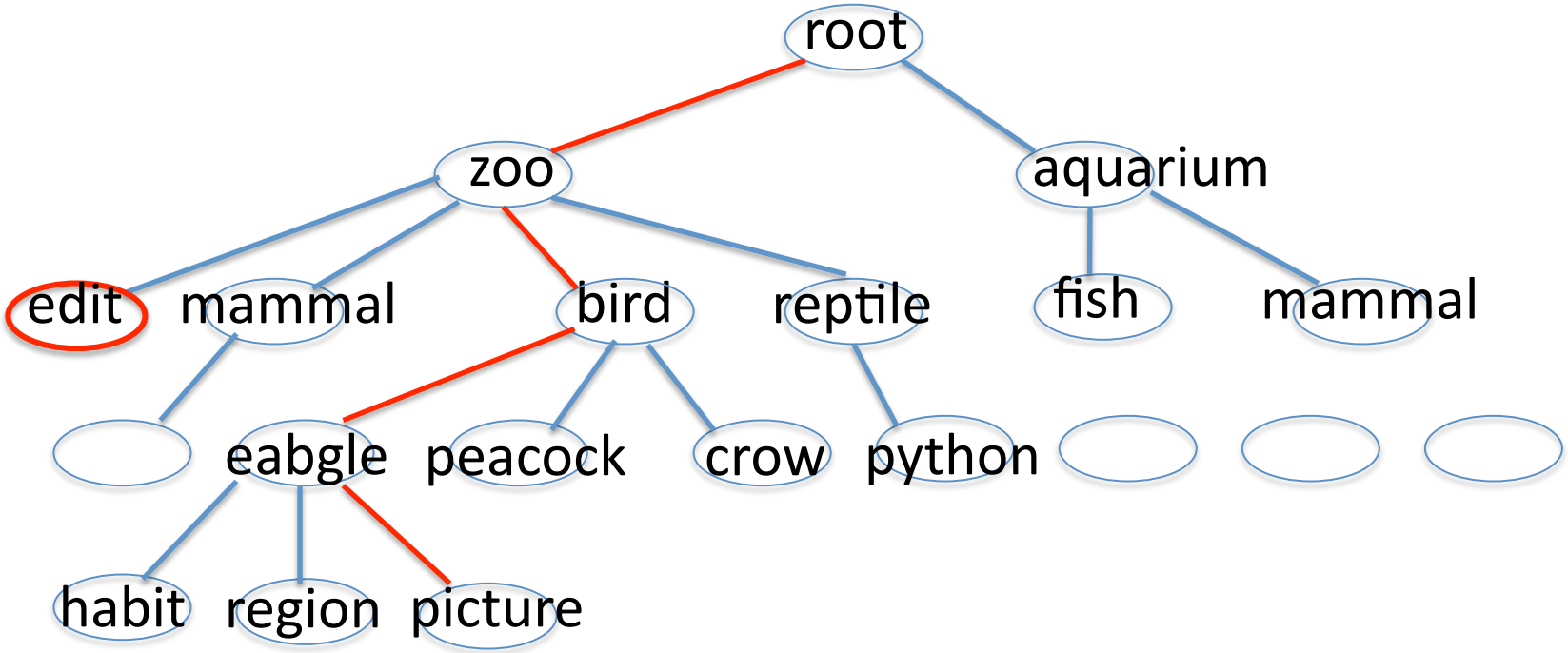
オブジェクトパブリッシング



1. パブリッシャが入力メッセージをZopeオブジェクトに変換
2. パブリッシャはURLを見て該当するオブジェクトを探す（トラバーサル）
3. うまく見つけたら、それにリクエストを渡し、戻ってきたものを返す。
4. 見るからなければ、エラーオブジェクトを返す。

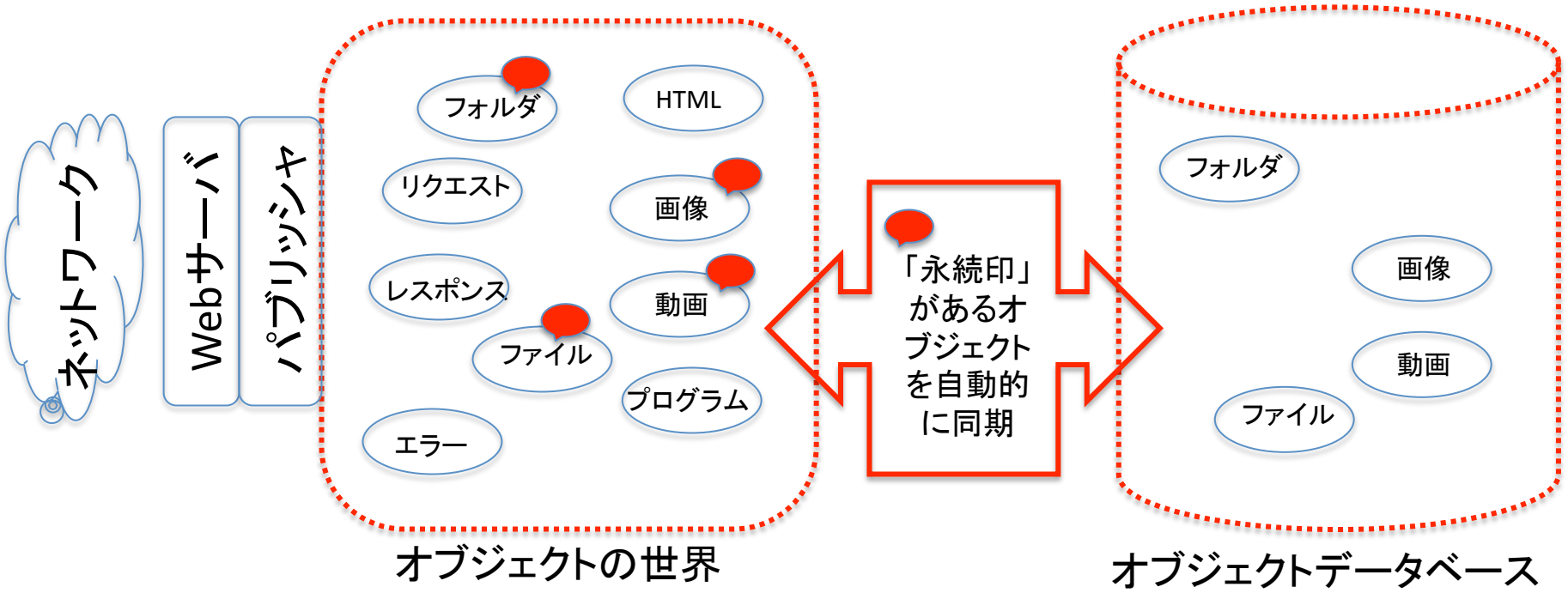
トラバーサルと獲得

http://www.mysite.com/zoo/bird/eagle/picture/edit



トラバーサル：スキーのジグザグ横切りのように探す
獲得：下になければ、上に戻って探す

オブジェクトデータベース(ZODB)



Zope 2は高い生産性で人気

TTW(Through-The-Web)での開発が人気
さらにCMF(Content Management Framework)により
CMS基盤としての地位を確立。

そして、

2001年 Jim Fultonとコミュニティによって、
Zope 3プロジェクト開始！

互換性を考慮せず、完全に新規開発！

なぜ？

他の人が作ったソフトを使う時の問題

- クラスやメソッドを呼ぶ時の名前が自分のアプリと違う
- クラスやメソッドの機能(API)がちょっと違う
- セキュリティの設定が違う
- メニュー構成が違う

Zope 3

- プログラムの再利用性の向上が目的
- 互換性を考慮しない完全な新規開発
- コンポーネントアーキテクチャの導入
- Zope 2の良い点を継承し、学んだ事を反映
- 最初からZope Corporationとコミュニティによる共同開発

プログラムの再利用性

一度書いたら、何度も使う

コピペして、改造するのは再利用とは言えない。
　改造のために理解する手間、改造する手間、テストする手間がかかる
　不具合の時に元プログラムが悪いのか改造が悪いのか不明瞭になる

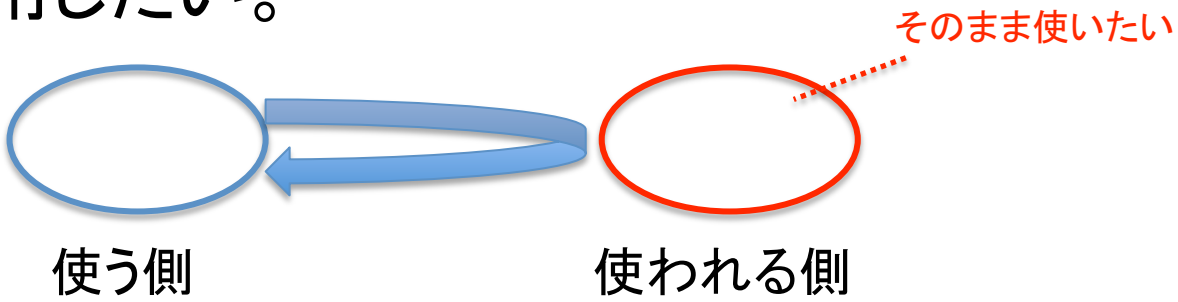
元のプログラムはいっさい変更しないで再利用する

コンポーネントアーキテクチャによる実装

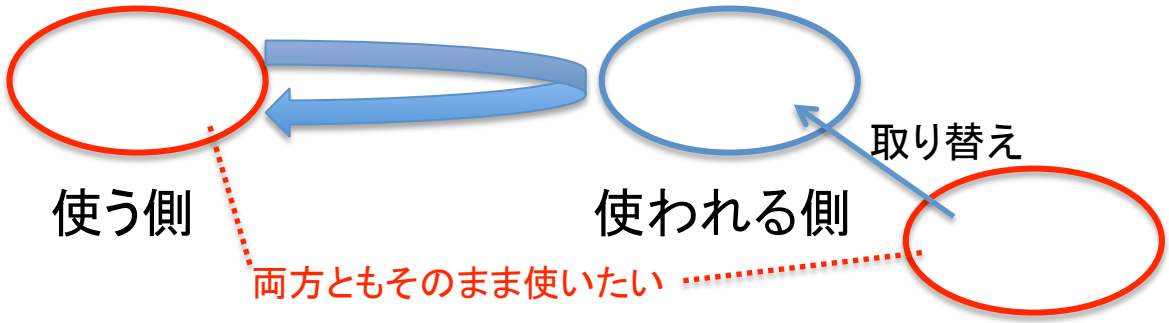
Zope 2を改造して実現するのは不可能

再利用って？

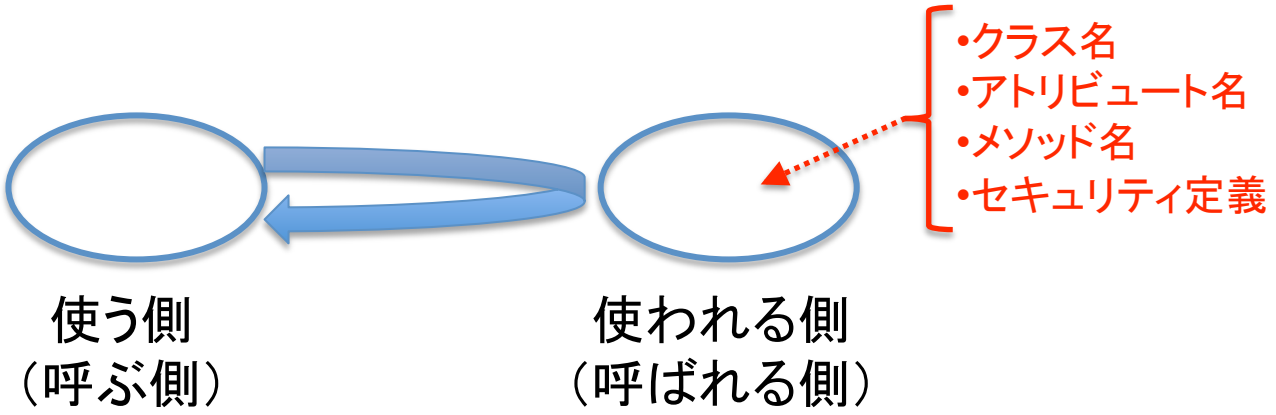
1. 新しいアプリケーションで既存プログラムを部品として再利用したい。



2. 既存アプリケーションで機能の一部分を別のプログラムで置換えたい。



再利用するのに何が問題？



コードの中に名前が入っているので、名前が違うとコードも変更しなくてはならない

コードの中にセキュリティ定義が入っているので、セキュリティが違うとコードも変更しなくてはならない。

コンポーネント
アーキテクチャ

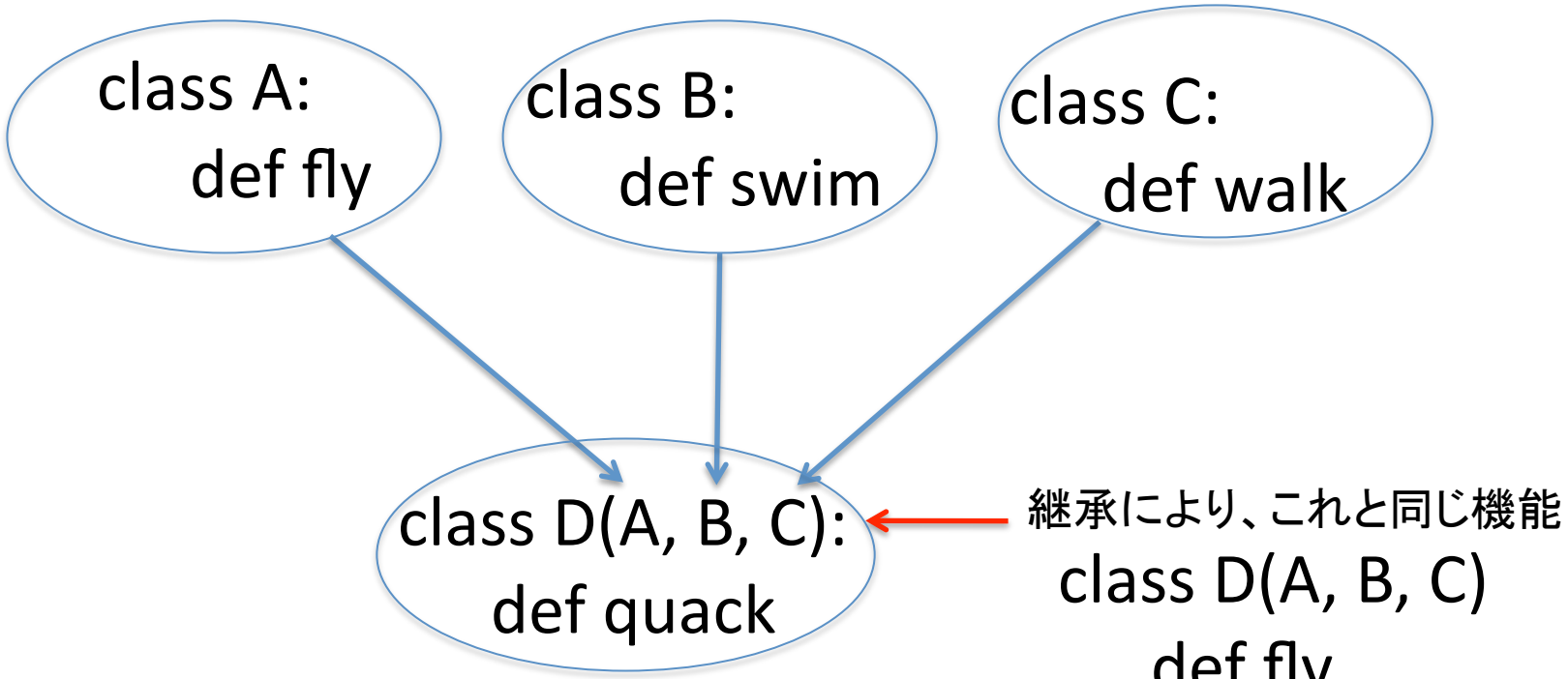
Componet Architecture

コンポーネントアーキテクチャ

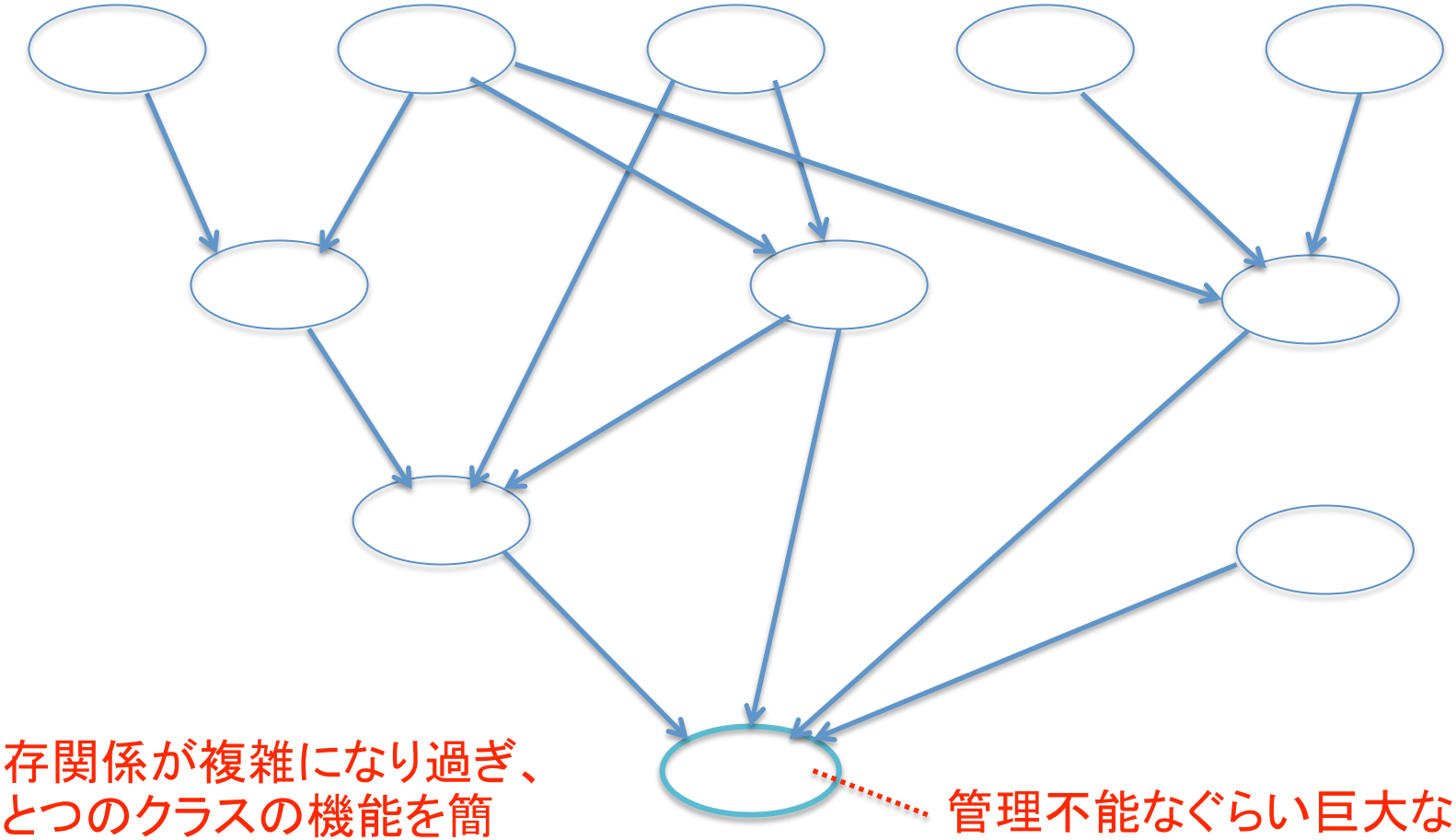
プログラムの再利用化のための考え方の体系

- サブクラス化よりも、委任を使う
- 再利用の単位として、コンポーネントというものを考える。
- コンポーネントのAPI定義を書いたインタフェースを作り、コンポーネントと結びつける。
- コンポーネントをインタフェースなどの特徴情報とともにコンポーネントレジストリに登録する。
- コンポーネントは、コンポーネントレジストリを通してインタフェースなどの特徴情報を基にして探し出される。
- 自分で書く部分と流用する既存コンポーネントの間で名前や機能が違う場合、アダプタで吸収する。
- セキュリティ定義をコンポーネントから取り除き、XML形式のZCMLで定義する。各パッケージのZCMLはアプリケーション側のZCMLでオーバーライドできる。

サブクラス化


class A:
def fly
class B:
def swim
class C:
def walk
class D(A, B, C):
def quack
継承により、これと同じ機能
class D(A, B, C)
def fly
def swim
def walk
def quack

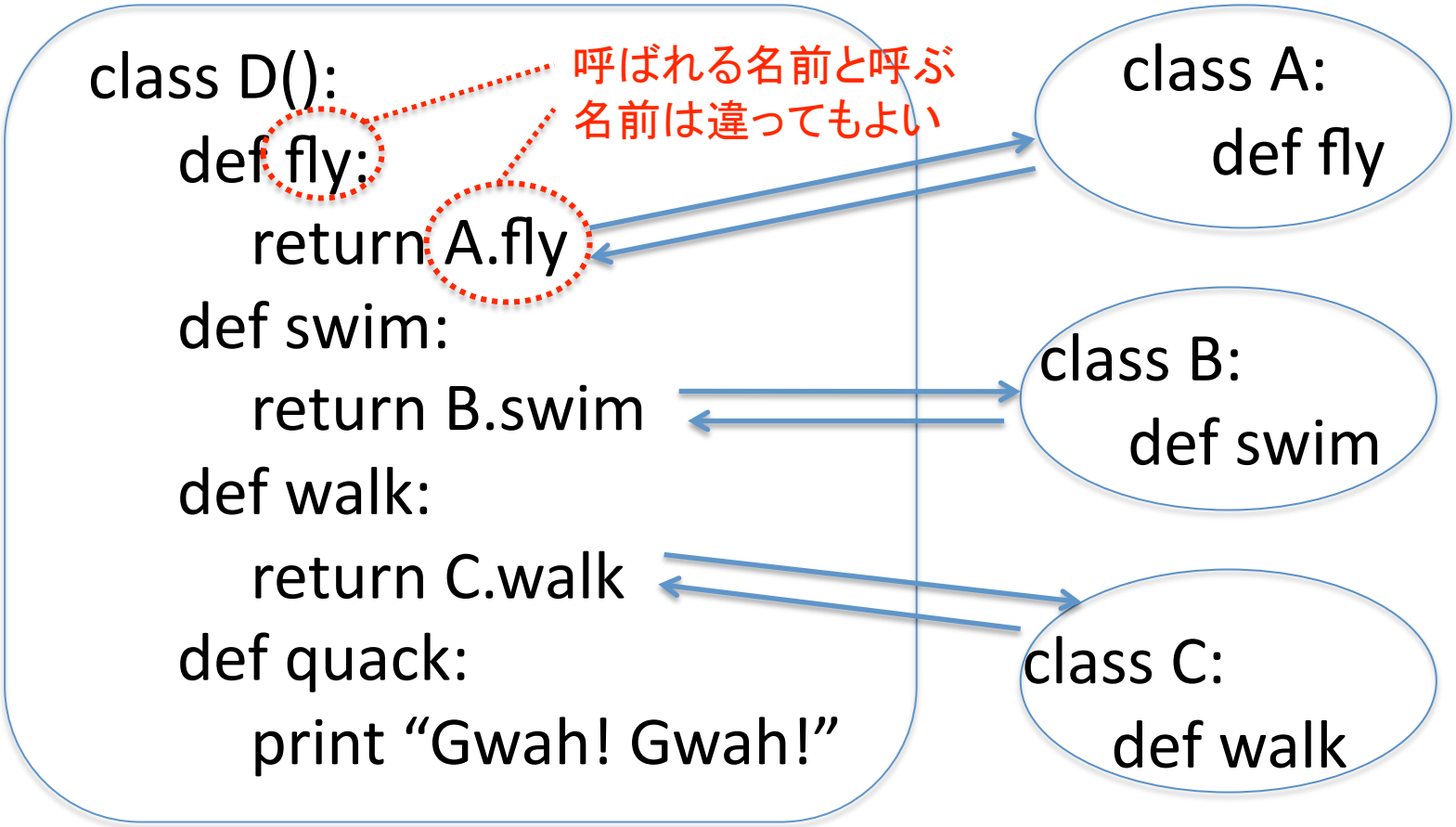
サブクラス化の使い過ぎ



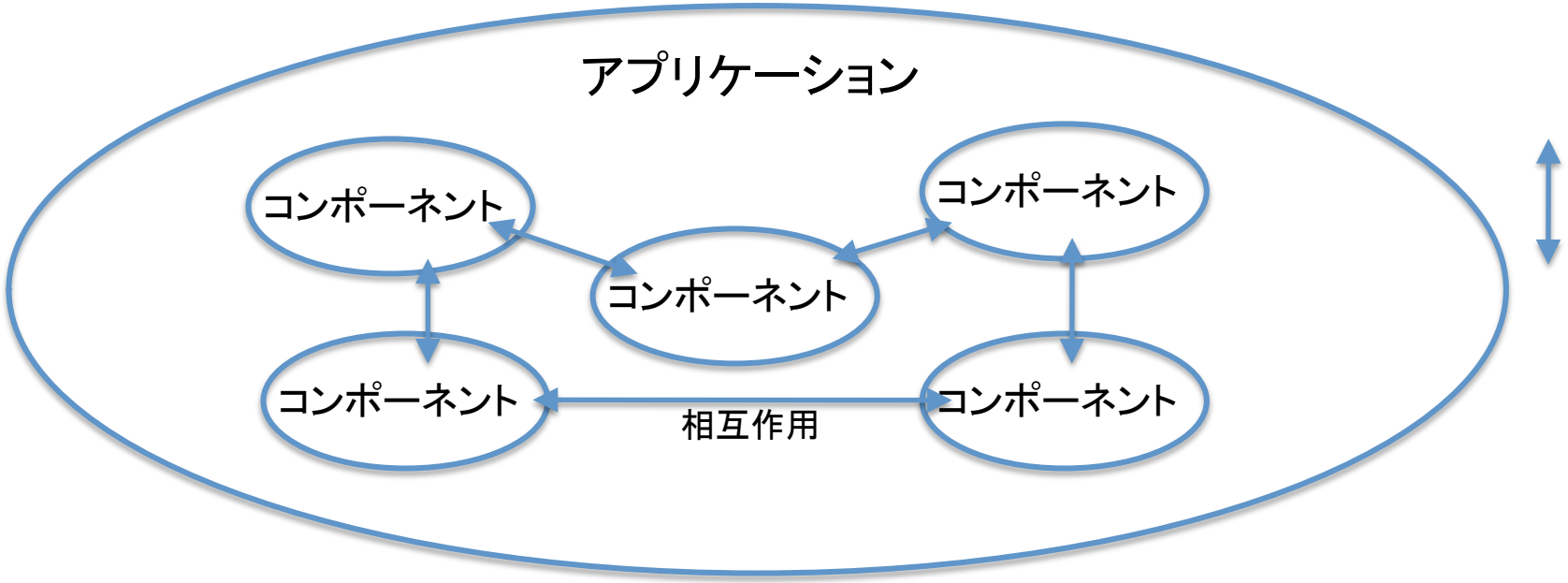
依存関係が複雑になり過ぎ、
ひとつのクラスの機能を簡
単には変更できない

管理不能なぐらい巨大なク
ラスができてしまう。
（たとえば、メソッド数が数百になり、メ
ソッド名の重複が発生）

委任


class D():
def fly:
return A.fly
def swim:
return B.swim
def walk:
return C.walk
def quack:
print “Gwah! Gwah!”
呼ばれる名前と呼ぶ
名前は違ってもよい
class A:
def fly
class B:
def swim
class C:
def walk

コンポーネント

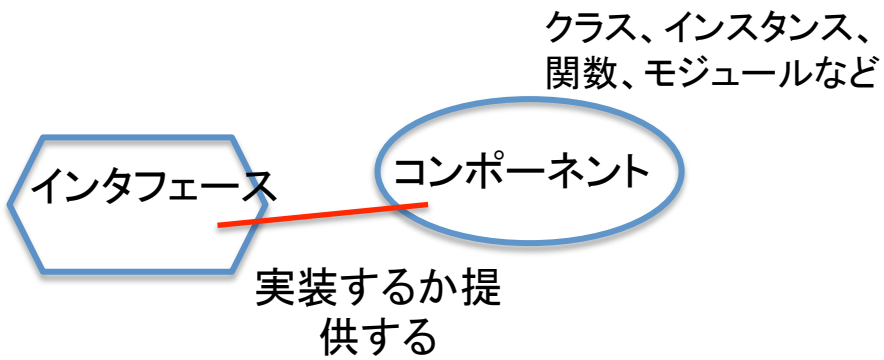


Zope 3が機能の塊として扱うオブジェクト
アプリケーションはコンポーネントの集まり
コンポーネント同士は相互作用する

インタフェース

Interface Object

インタフェース



- インタフェースはコンポーネントのAPIを記述した新しいタイプのオブジェクト
- 記述方法はclass文, def文, =を使を使うので、見かけはクラス定義に似ている
- でも、”Interface”を拡張するので、できるものは、インタフェースオブジェクト
- 書かれたものは実行できないし、アトリビュート参照できない。
- 名前は必ず”I”（大文字のアイ）で始まる
- インタフェースを実装したり、提供したりするものがコンポーネント

インタフェースのコード

- コード例

頭文字がI（アイ）

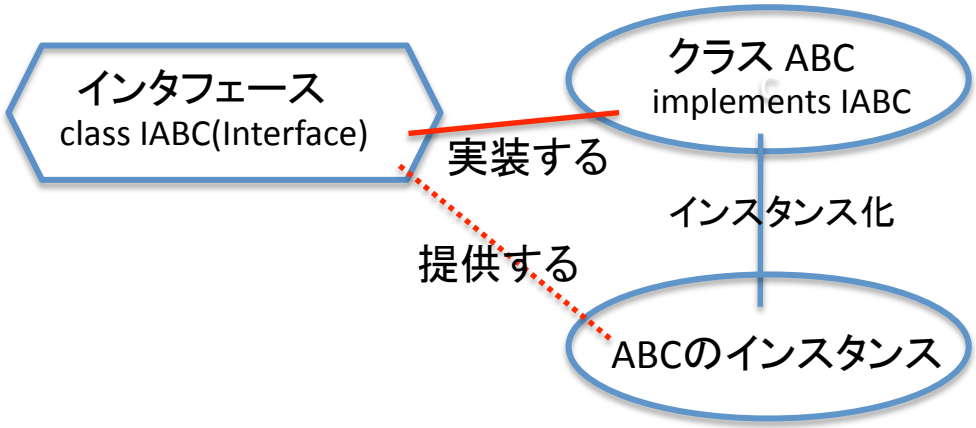
“Interface”を拡張

```
class ILocaleInheritance(Interface):
    """Locale inheritance support.

    Locale-related objects implementing this interface are able to ask for its
    inherited self. For example, 'en_US.dates.monthNames' can call on itself
    'getInheritedSelf()' and get the value for 'en.dates.monthNames'.
    """

    __parent__ = Attribute("The parent in the location hierarchy")

    __name__ = TextLine(
        title = u"The name within the parent",
        description=u"""The parent can be traversed with this name to get
                       the object.""")

    def getInheritedSelf():
        """Return itself but in the next higher up Locale."""
```

インタフェース(API)の記述

インタフェース

- コンポーネントとの関係

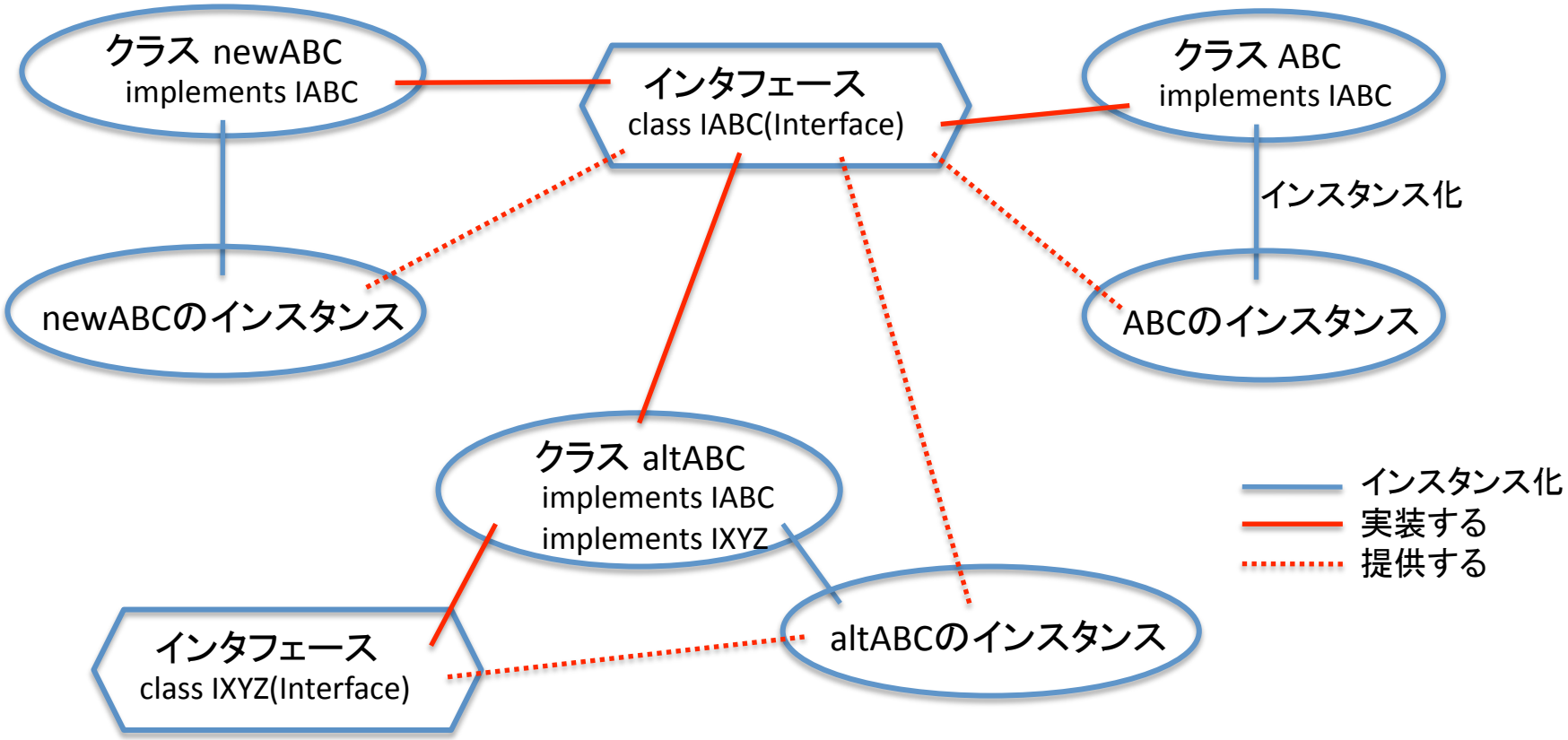



```
class Inheritance(object):
    """A simple base version of locale inheritance.

    This object contains some shared code amongst the various
    'ILocaleInheritance' implementations.
    """

    implements(ILocaleInheritance)

    # See zope.i18n.interfaces.locales.ILocaleInheritance
    __parent__ = None

    # See zope.i18n.interfaces.locales.ILocaleInheritance
    __name__ = None

    def getInheritedSelf(self):
        """See zope.i18n.interfaces.locales.ILocaleInheritance"""
        if self.__parent__ is None:
            raise NoParentException('No parent was specified.')
        parent = self.__parent__.getInheritedSelf()
        if isinstance(parent, dict):
            return parent[self.__name__]
        return getattr(parent, self.__name__)
```

ここで実装

インタフェース

- 複数のコンポーネントが同じインタフェースを持てる
- コンポーネントは複数のインタフェースを持てる



インタフェースの役割

- APIの記述

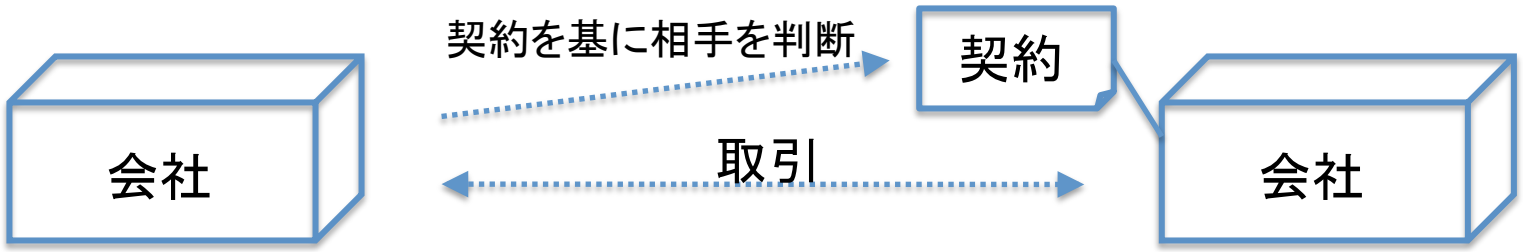
- コンテンツスキーマの記述
  - スキーマはインタフェースの一種

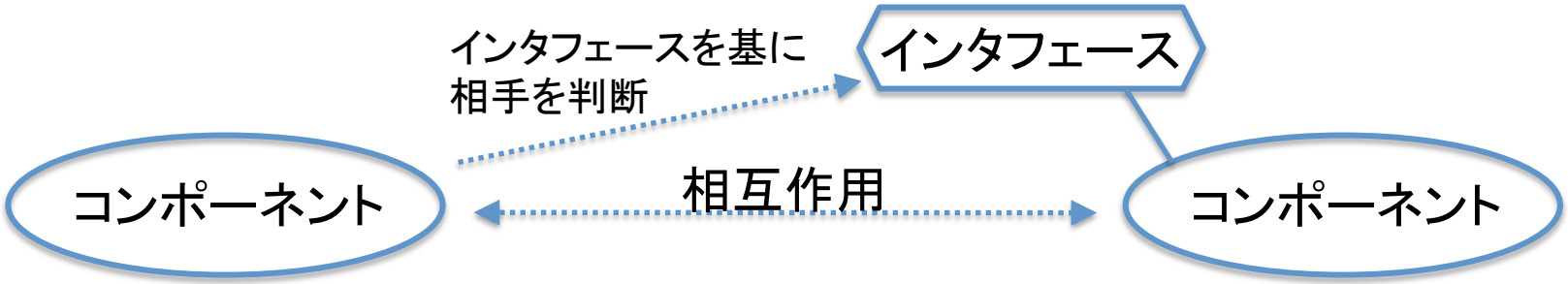
- マーカインタフェース（種類を識別する標識）
  - コンポーネントを、それが提供しているインタフェースで見分ける
  - 一つのコンポーネントは、複数のインタフェースを持てる
  - APIが何も書かれない抽象インタフェースもある

- ドキュメントとして
  - コンポーネントが何かを知るのに、これを見ればたいていは十分

たとえて言うなら

•コンポーネントを会社に見立てれば、インタフェースは契約





で、結局のところ

Zope 2では、相手を相手が決めた名前で呼ぶ

```
x = ABC()
```

Zope 3では、相手をインタフェースで呼ぶ

```
x = getUtilities(IABC)
```

レジストリにこのインタフェースで登録されたコンポーネントが返される

あるいは、コンフィグ(ZCML)で定義した名前で呼ぶ

```
x = getUtilities(IABC, name=u'myABC')
```

レジストリにこのインタフェースとコンポーネント名で登録されたコンポーネントが返される

アダプタ

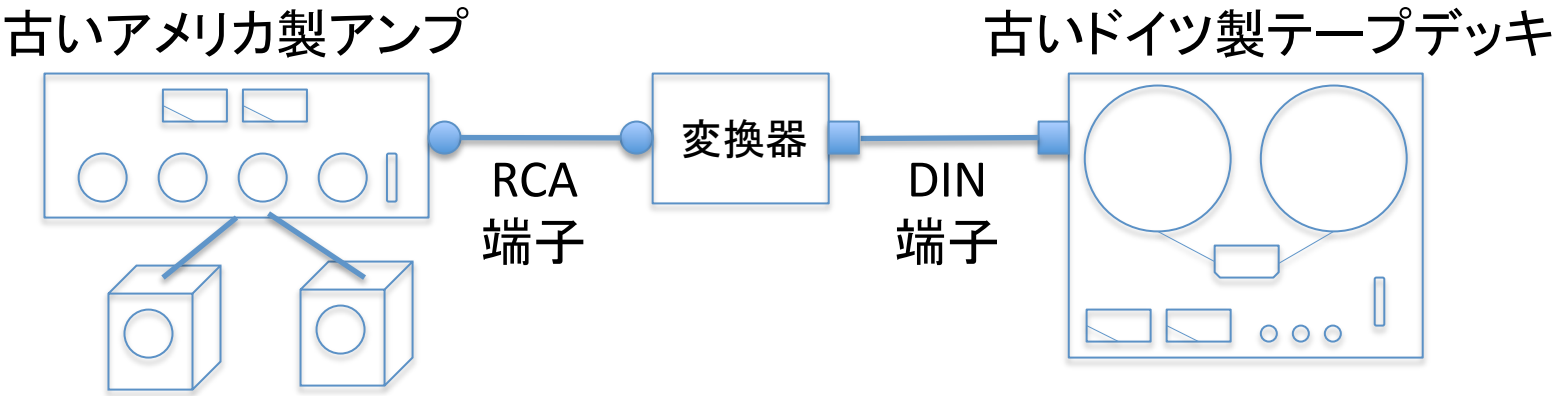
Adapter

アダプタ

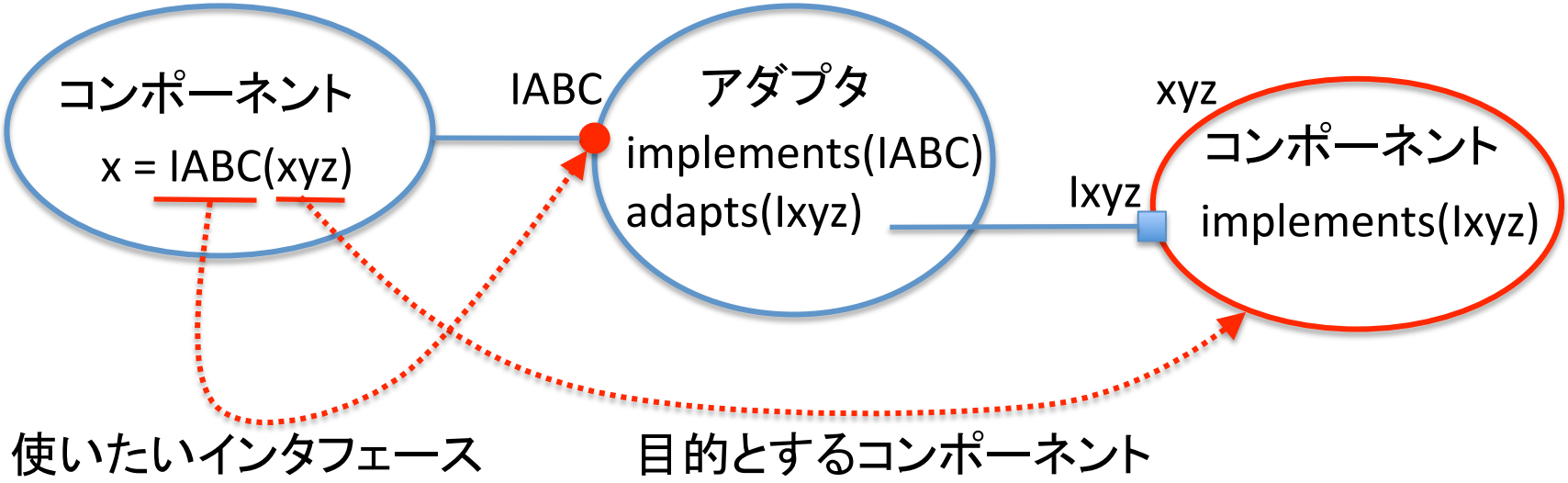
インタフェースの違いを吸収するための仲介コンポーネント

たとえるなら、

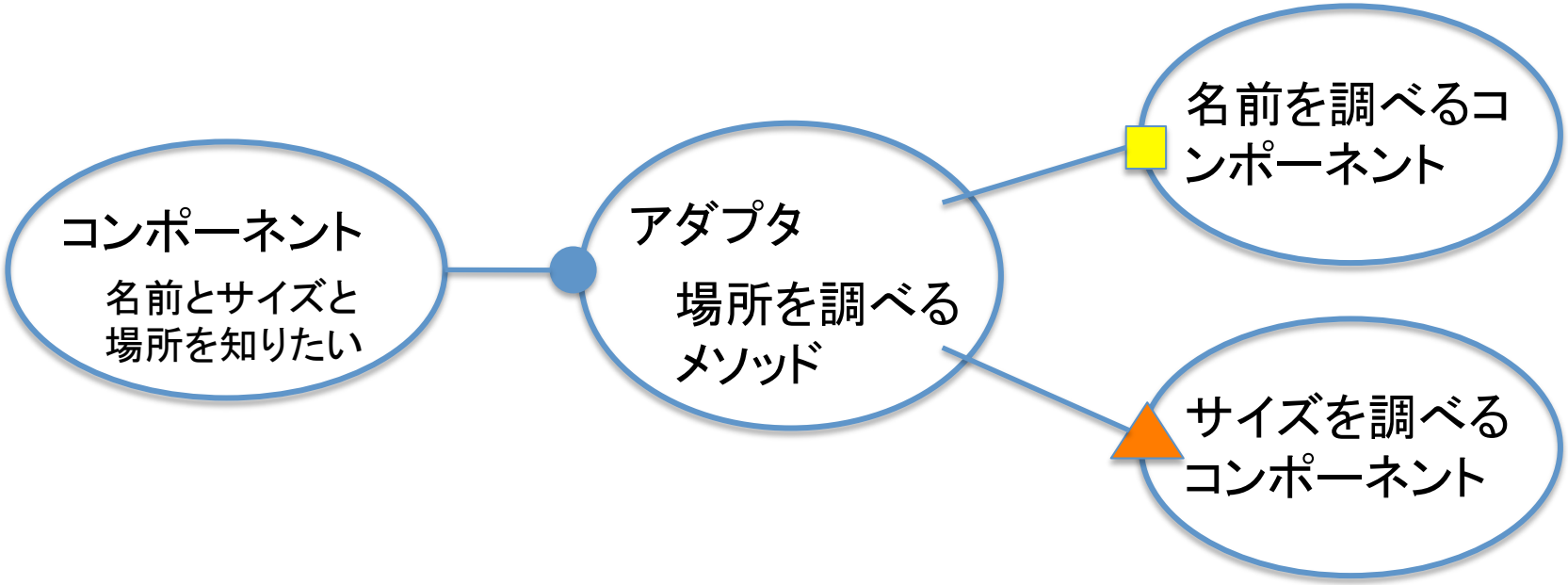
アダプタがインタフェースの違いを吸収



アダプタの使用


コンポーネント
x = IABC(xyz)
IABC
アダプタ
implements(IABC)
adapts(Ixyz)
Ixyz
xyz
コンポーネント
implements(Ixyz)
使いたいインタフェース
目的とするコンポーネント

たとえば



インタフェースの違いを吸収しながら、各種コンポーネントの機能を組み合わせて提供する

ZCML

Zope Configuration
Management Languate

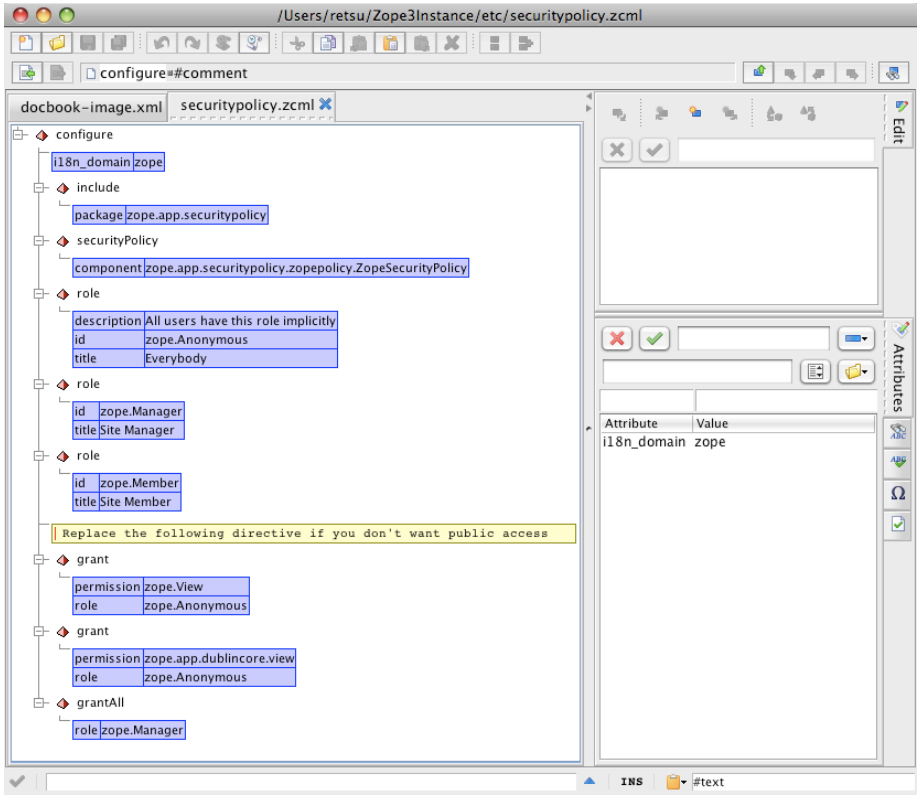
ZCML

- XML形式のコンフィグレーション管理言語
- そのファイルもZCMLとよばれる
- Pythonコード内から以下の機能を取り除く
  - コンポーネントの登録
  - セキュリティ構成定義
  - メニュー構成定義
  - APIドッック（コード内ドキュメント）の登録
- アプリケーション側から追加やオーバーライドが可能

ZCML

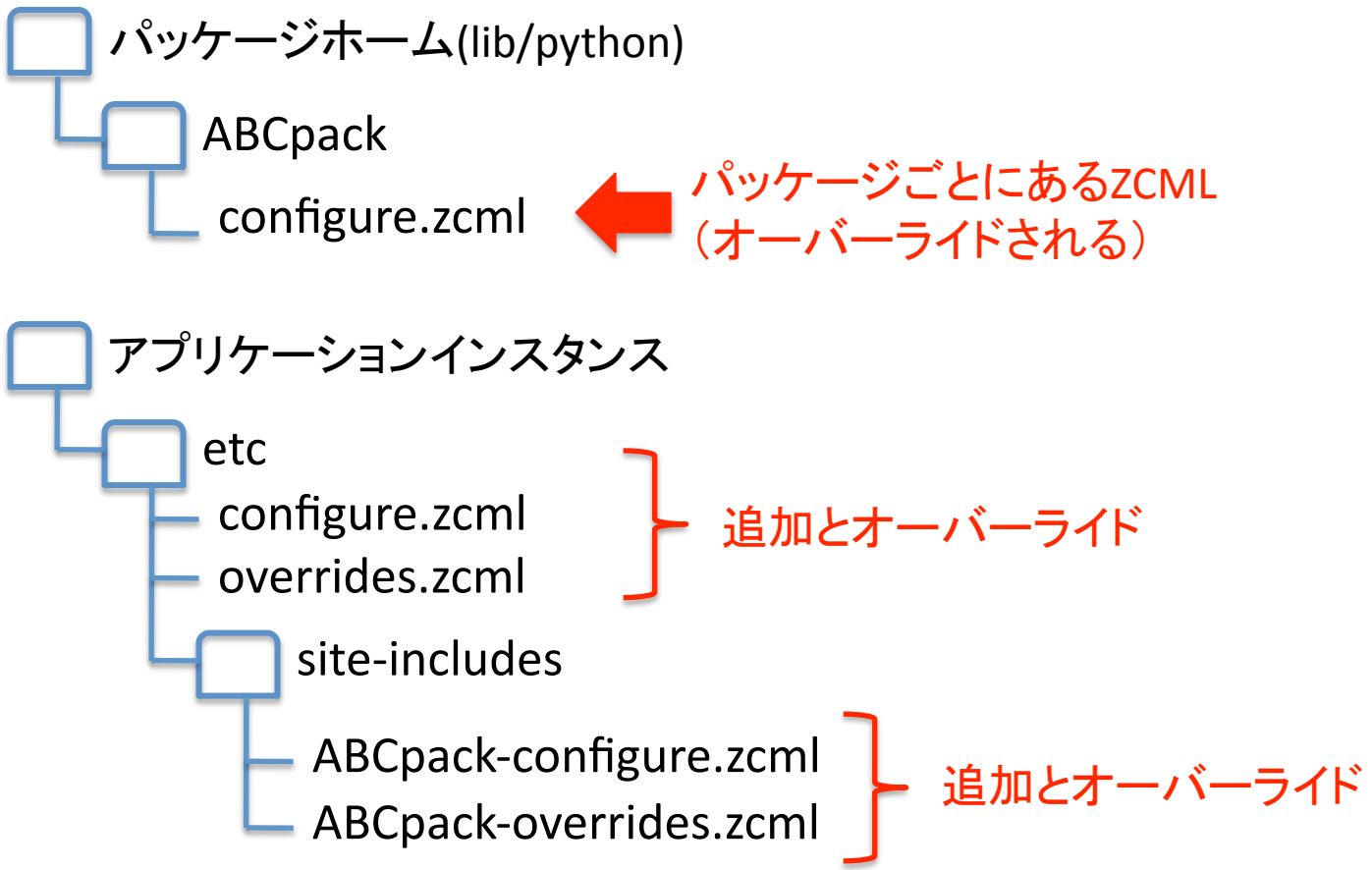
- XML形式のZope Configuration Management Language
  （Zopeコンフィグレーション管理言語）
- XMLなので、通常のXMLエディタや解析ツールが使える

Pythonを知らなくても、
コンフィグできる！



ZCMLの追加とオーバーライド

ロジックコードの中ではなく、ZCMLファイルに記述
パッケージごとのZCMLは、インスタンスで追加とオーバーライド可能



セキュリティ

Security

セキュリティ定義

•Zope 3ではセキュリティ定義をコードのではなく、ZCMLとしてコンフィグの中で定義

•各パッケージのZCMLは、パッケージの外側からZCMLを追加したりオーバーライドすることが可能

•その結果、パッケージをまったく変更することなくセキュリティ定義を変更できる

Zope 2のセキュリティ定義

コードの中に記述

```
#   'portal_membership' interface methods
#
security.declareProtected( ListPortalMembers, 'getRoster' )
def getRoster(self):
    """ Return a list of mappings for 'listed' members.

    If Manager, return a list of all usernames.  The mapping
    contains the id and listed variables.
    """
    isUserManager = _checkPermission(ManageUsers, self)
    roster = []
    for member in self.listMembers():
        if isUserManager or member.listed:
            roster.append({'id':member.getId(),
                           'listed':member.listed})
    return roster

security.declareProtected(ManagePortal, 'setMembersFolderById')
def setMembersFolderById(self, id=''):
    """ Set the members folder object by its id.
    """
    self.membersfolder_id = id.strip()
```

パーミッション定義

パーミッション検査

パーミッション定義

Zope 3のセキュリティ定義

ロジックコードの中ではなく、ZCMLファイルに記述

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

 <class class=".onlinehelp.OnlineHelp">
  <require
     permission="zope.View"
     interface=".interfaces.ISourceTextOnlineHelpTopic"
     />
  <require
     permission="zope.View"
     attributes="context"
     />
 </class>
```

パーミッション定義

パーミッション定義

ここまでの、まとめ

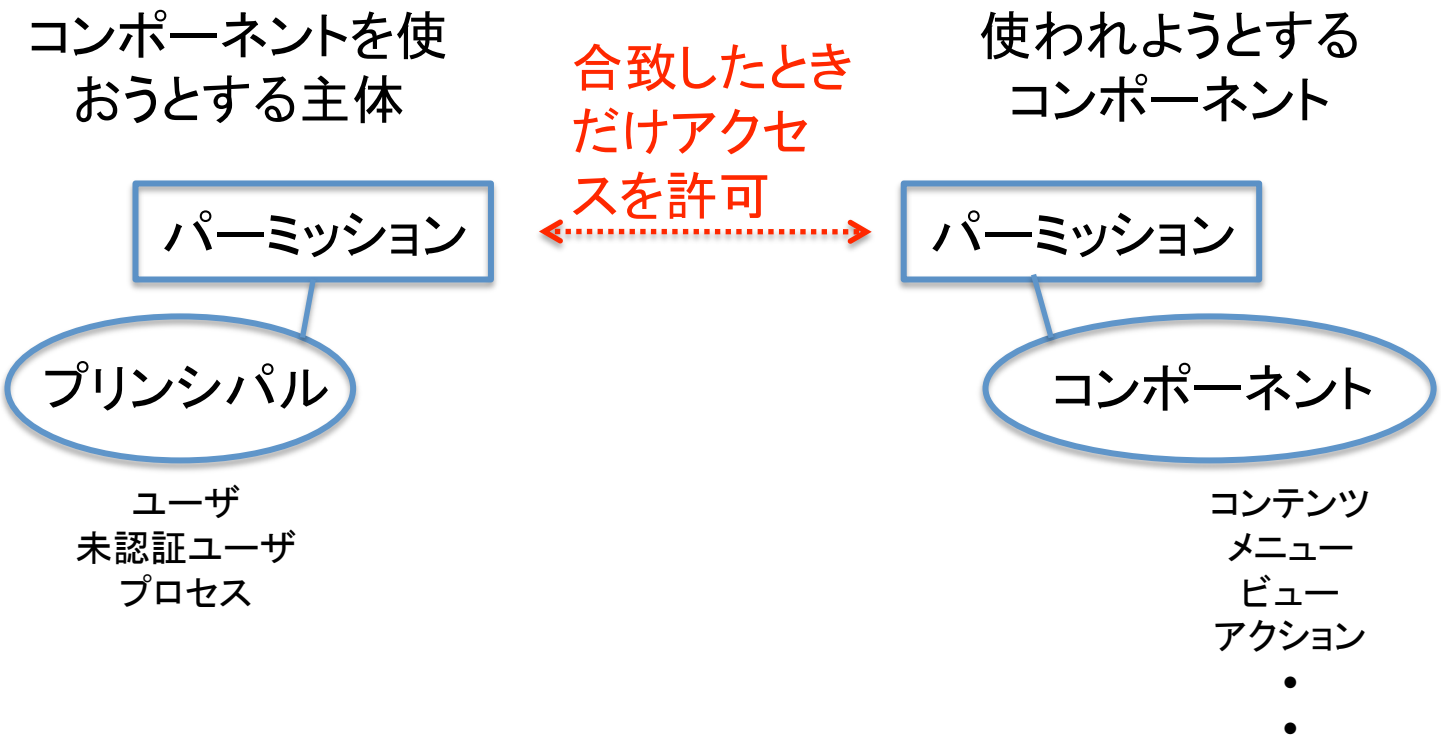
•インタフェースやZCMLで定義したコンポーネント名を使ってレジストリ経由で呼び出す。こうすれば、使う側のコンポーネントは使われるコンポーネントの名前を直接的に使わずに済む。

•目標オブジェクトを使いたいインタフェース経由で呼び出すことにより、レジストリに登録されたアダプタが呼び出される。そのアダプタがAPIの違いを吸収することにより、呼び出す側呼び出される側ともにコードを変更せずに済む。

•セキュリティとメニュー定義をコンポーネント内ではなく、ZCMLを使ったコンフィグレーションで定義する。そのZCMLはアプリケーション側が定義したZCMLでオーバーライドできる。

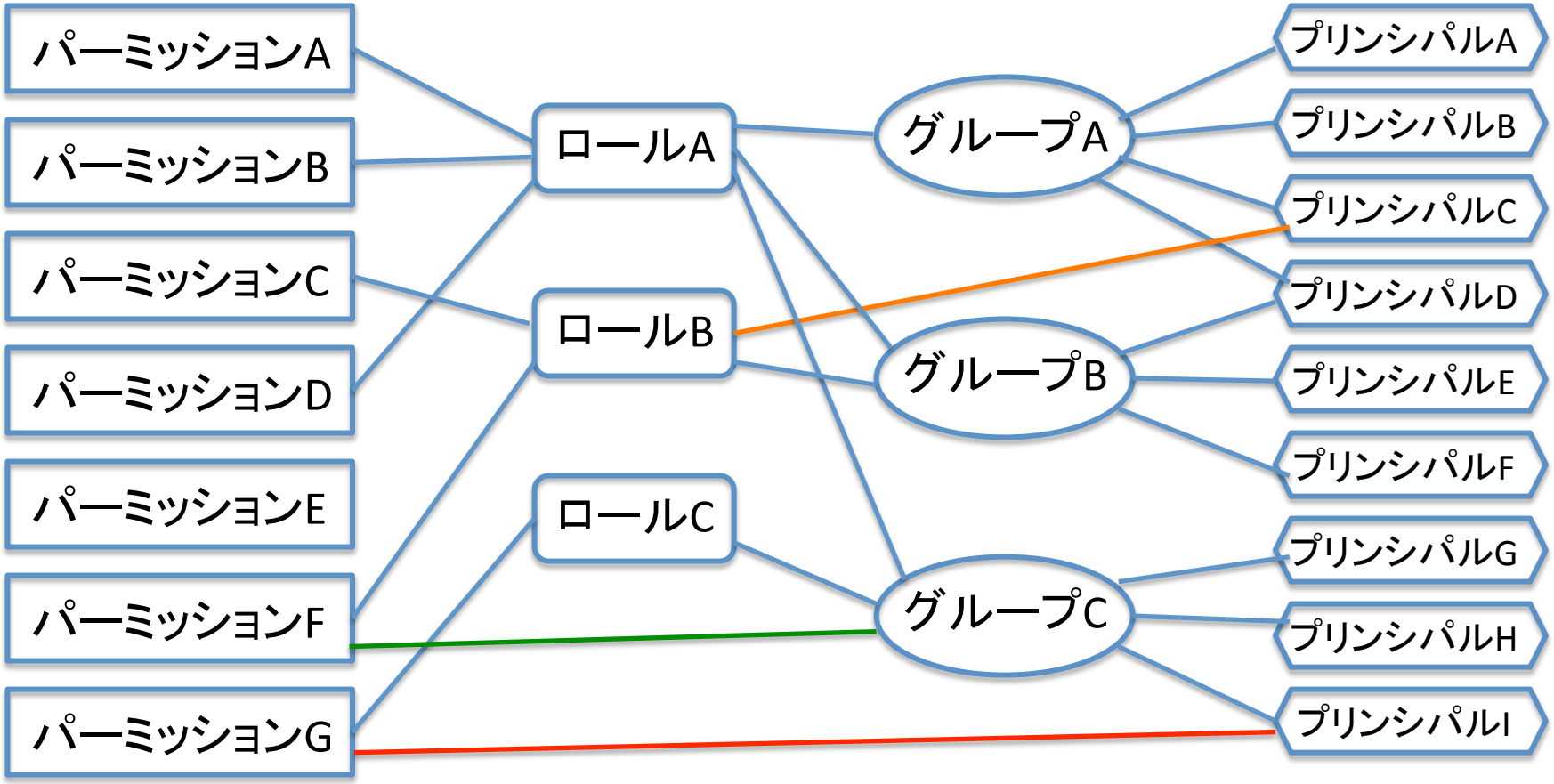
セキュリティモデル

基本的な機構はパーミッションベース



ロールとグループ

パーミッションは数百にもなる。とてもそのままでは管理不能
ロールとグループを使ってパーミッションを束ねて管理
さらに、ロールとグループはそれぞれ階層化可能



セキュリティ

セキュリティはコンポーネントの中ではなく、ZCMLで定義する。
つまり、セキュリティはプログラマでなくても定義できる。

プリンシパル側

```
<configure xmlns='http://
namespaces.zope.org/zope'>

 <unauthenticatedPrincipal
  id="zope.anybody"
  title="Unauthenticated User" />

 <unauthenticatedGroup
  id="zope.Anybody"
  title="Unauthenticated Users"
  />

 <authenticatedGroup
  id="zope.Authenticated"
  title="Authenticated Users"
  />
```

コンポーネント側

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

 <class class=".onlinehelp.OnlineHelp">
  <require
     permission="zope.View"

interface=".interfaces.ISourceTextOnlineHelp
Topic"
     />
  <require
     permission="zope.View"
     attributes="context"
     />
 </class>
```

セキュリティポリシー

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

  <include package="zope.app.securitypolicy" />

  <securityPolicy

component="zope.app.securitypolicy.zopepolicy.
ZopeSecurityPolicy" />

  <role id="zope.Anonymous" title="Everybody"
          description="All users have this role
implicitly" />
  <role id="zope.Manager" title="Site Manager" /
>
```

メニュー構成の定義

メニュー構成

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

 <menu
    id="zmi_views"
    title="Views"
    description="Menu for displaying alternate representations of an object"
    />

 <menu
    id="zmi_actions"
    title="Actions"
    description="Menu for displaying actions to be performed"
    />

 <menu
    id="zope.app.container.add"
    title="Add"
    description="Menu for objects to be added according to
                 containment constraints"
    interface="zope.app.publisher.interfaces.browser.AddMenu"
    />
```

品質（テスト）

Quality Assurance
(Testing)

品質（テスト）

多様なテストの種類とサポート

- ユニットテスト
- インテグレーションテスト
- ファンクショナルテスト

テストランナーによるテスト作業の自動化

- テスト
- テストケース
- テストスイート

テストファーストの文化

コードを書いても、テストも書いてパスしないとコミットできない

ドックテスト

ドキュメントとしても読めるテスト記述

event.txt

```
Events
======

The Component Architecture provides a way to dispatch events to event
handlers.  Event handlers are registered as *subscribers*
a.k.a. *handlers*.

Before we can start we need to import ``zope.component.event`` to make
the dispatching effective:

  >>> import zope.component.event

Consider two event classes:

  >>> class Event1(object):
  ...     pass

  >>> class Event2(Event1):
  ...     pass

Now consider two handlers for these event classes:

  >>> called = []
```

テストとして実行される

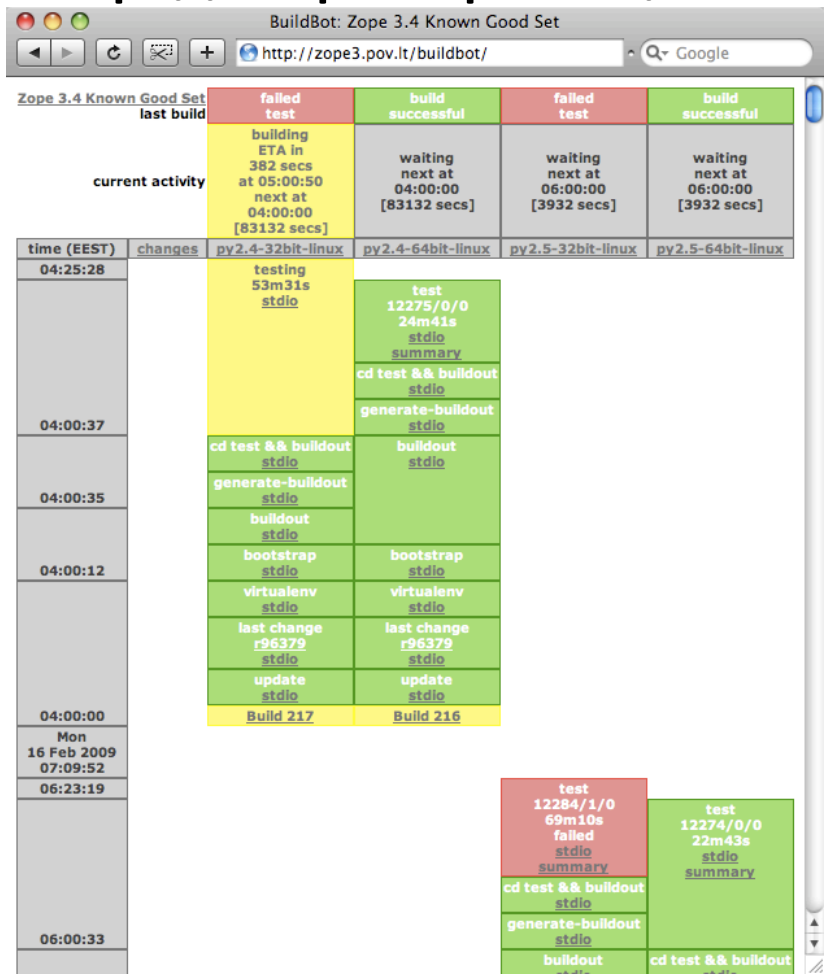
テストとして実行される

テストとして実行される

ドキュメントとして
開発者が読む

Zope 3ビルドアウトテスト

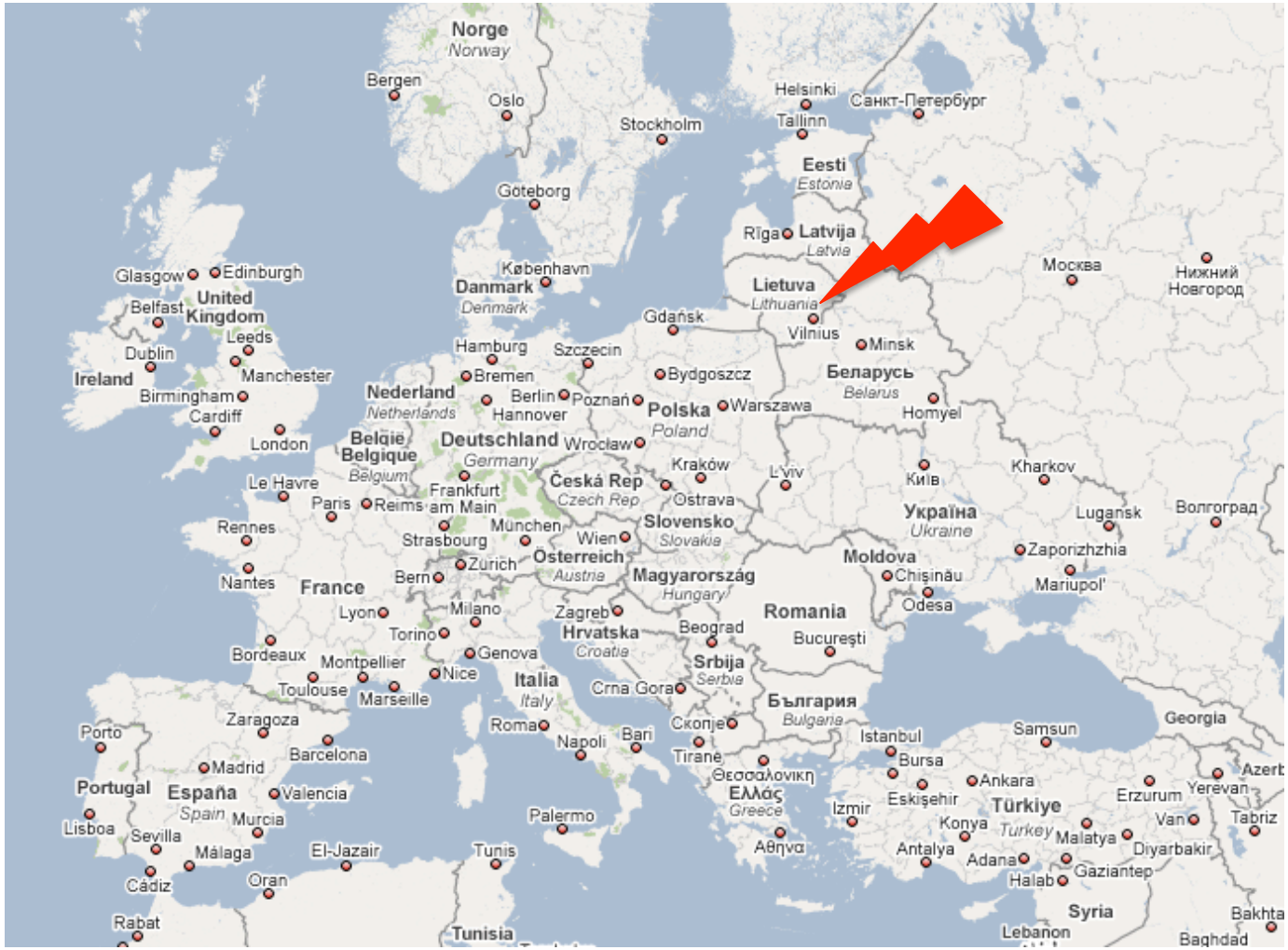
http://zope3.pov.lt/builtbot/




```
  Ran 103 tests with 0 failures and 0
errors in 0.967 seconds.  Tear down
zope.testbrowser.tests.TestBrowserLayer in
0.002 seconds.Running
zope.traversing.tests.layer.TraversingLayer
tests:  Running in a subprocess.  Set up
zope.traversing.tests.layer.TraversingLayer
in 0.668 seconds.  Running:.......  Ran 7
tests with 0 failures and 0 errors in 0.169
seconds.  Tear down
zope.traversing.tests.layer.TraversingLayer
in 0.001 seconds.Total: 12275 tests, 0
failures, 0 errors in 24 minutes 22.033
seconds.
```

1万2千個以上のユニットテストが毎日何回も行なわれている。
さらにこれとは別に、各インテグレータは自分の環境でテストしている。

リトアニアってどこ？


Norge
Norway
Bergen
Oslo
Stockholm
Helsinki
Санкт-Петербург
Tallinn
Eesti
Estonia
Göteborg
Rīga
Latvija
Latvia
København
Danmark
Denmark
Lietuva
Lithuania
Vilnius
Gdańsk
Москва
Нижний Новгород
Glasgow
Edinburgh
Belfast
United Kingdom
Leeds
Dublin
Ireland
Manchester
Birmingham
Cardiff
London
Hamburg
Szczecin
Bremen
Nederland
Netherlands
Berlin
Poznań
Hannover
Bydgoszcz
Polska
Warszawa
Poland
Minsk
Беларусь
Belarus
Homyel
Belgie
Belgique
Belgium
Deutschland
Germany
Wrocław
Kraków
L'viv
Київ
Kharkov
Le Havre
Paris
Reims
Frankfurt am Main
Česká Rep
Czech Rep
Ostrava
Slovensko
Slovakia
Україна
Ukraine
Lugansk
Волгоград
Rennes
München
Strasbourg
Wien
Zaporizhzhia
Nantes
France
Zürich
Österreich
Austria
Magyarország
Hungary
Moldova
Chişinău
Mariupol'
Bern
Odesa
Lyon
Milano
Zagreb
Romania
Torino
Hrvatska
Croatia
Beograd
București
Bordeaux
Montpellier
Genova
Nice
Srbija
Serbia
Toulouse
Marseille
Italia
Italy
Crna Gora
България
Bulgaria
Zaragoza
Roma
Napoli
Bari
Скопје
Georgia
Porto
Barcelona
Madrid
Tirane
Θεσσαλονίκη
Istanbul
Samsun
Bursa
Ankara
Azerb
Portugal
España
Valencia
Spain
Ελλάς
Greece
Eskişehir
Erzurum
Yerevan
Lisboa
Murcia
Palermo
İzmir
Türkiye
Turkey
Konya
Van
Tabriz
Sevilla
Málaga
El-Jazair
Tunis
Αθηνα
Antalya
Malatya
Diyarbakir
Adana
Gaziantep
Cádiz
Oran
Halab
Syria
Rabat
Tunisia
Lebanon
Bakhta
Baghdad

その他もろもろ

etc……

その他の機能

- HTML/XMLテンプレーティング(ZPT)
- スキーマ、フォーム生成、バリデーション
- ダブリンコア準拠のメタデータ
- 国際化と地域化
- インデックス化、カタログ、検索
- ドックテスト（テストとドキュメントの共用化）
- テスト作業の自動化
- セルフドキュメンテーション（コード内記述とそのブック化）
- ビルドアウト（デプロイメントの自動化），KGS(Know Good Set)
- WSGI準拠（他Webサーバーフレームワークとの連係）
- ZEO, ZRSによるスケーラビリティ（負荷分散、ZODBのクラウド化）
- リレーショナルデータベース接続，ORマッパー
- WfMC準拠のワークフローエンジン、XPDLのサポート
- FTP, WebDAV, XML-RPC接続
- Zope Foundation & コミュニティ
    -
    -
      まだまだたくさん！

開発過程の例

	コンポーネント開発				コンフィグレーション		
	設計	ロジック	デザイン	テスト	メニュー構成	セキュリティ構成	
使用言語	Python	Python, JavaScriptなど	HTML, XML, TAL, METAL, CSS	Pythonなど、テストツールに準じる	ZCML	ZCML	ZCML
実施者	システムアーキテクト	Pythonプログラマ	デザイナ	品質管理担当	導入担当	システム側セキュリティ担当	ユーザ側セキュリティ担当
作業内容	•アプリケーション全体の構成を決め、インタフェースを定義する	•インタフェースに沿ってコンポーネントを開発する	•デザインツールなどを使って画面のレイアウトや見栄えを作る	•ユニットテストやインテグレーションテストを行なう	•他コンフィグ担当者と協議しながらメニューを定義する	•他コンフィグ担当者と協議しながらシステム側セキュリティを定義する	•他コンフィグ担当者と協議しながらユーザ側セキュリティを定義する
成果物	•スキーマ定義 •API定義（どちらもインタフェース） •テスト定義	•コンポーネント(Pythonパッケージ)	•ページテンプレート(TAL, METALを埋め込んだHTML, XML) •CSS • グラフィックス(アイコン、ロゴ、バナーなど)	•テスト結果 •状況報告	•メニュー定義ZCML	•セキュリティポリシー定義ZCML •コンポーネントセキュリティ定義ZCML	•プリンシパルセキュリティ定義ZCML

プリンシ
パル定
義ZCML

Zope 3を使ったWebアプリケーション

http://www.zoomer.de

大規模な新聞社のサイト
18台のサーバー群
編集側ではZope4Mediaを使用
AJAX+JSONによるコメント機能

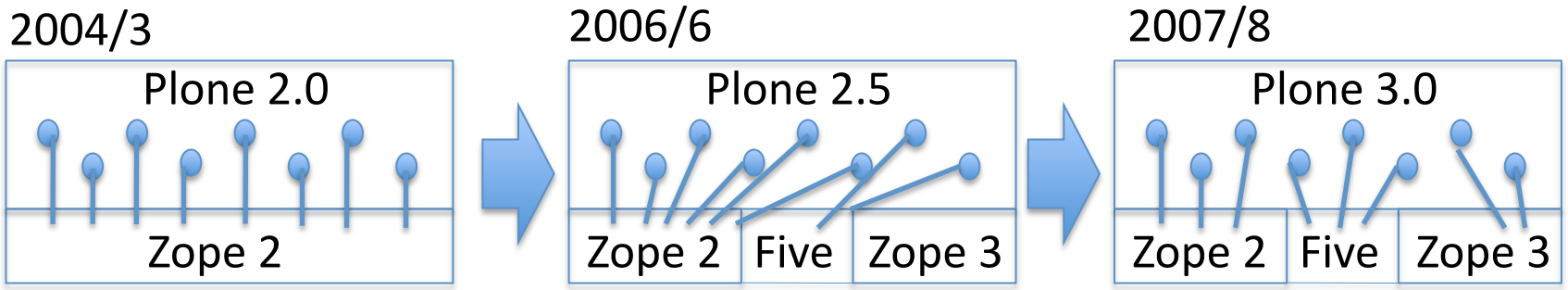
Zope 3とは？

- 自由なソフトウェアのコレクション（収集）であり、
- Zope Corporationとソフトウェア開発者の大きなコミュニティによって共同で開発され、
- 全体でも使え、一部だけでも使え、
- ソフトウェアコンポーネントを一緒に糊付けして使う際の複雑さを上手に扱い、
- Webや他のシステム上にオブジェクトを安全にパブリッシュし、
- 品質保証を容易してくれる。

Steve Alexander, EuroPython 2004にて

もう使っているかも？

- Ploneは”Five”によってどんどんZope 3の技術を使っています
- Plone/Zope-2.10.7-final-py2.4/lib/python/zope/ の中がZope 3です
- Plone/buildout-cache/eggs/ の中のProducts.で始まるもの以外がそうです
- Zope 3の特徴であるinterfaces.pyやconfigure.zcmlがすでにたくさんある



”Five”は Zope 2の中でZope 3の技術を使えるようにする技術
Zope 2.8以降はZope 3とFiveがZope 2に標準で含まれている

Zope 3をもっとよく知るには

グローバルコミュニティ

http://www.zope.org（メニュー左下にZope 3あり）

日本のコミュニティ

日本Zopeユーザー会：http://www.zope.jp
同メーリングリスト　：http://ml.zope.jp

Plone研究会　　：http://www.plone.jp
同メーリングリスト：http://ml.plone.jp

4階でデスク出してます！

トレーニングやビジネス的な問い合わせ

iccm inc.: info@iccm.jp

Zope/Plone/ERP5のブローカレッジ＆コンサルティング

良書

Web Component Development with Zope 3

3rd Edition

Philipp von Weitershausen

Springer

ISBN: 978-3-540-76447-2

日本語版　この春発売！