Programowanie i projektowanie obiektowe
XML

Paweł Daniluk

Wydział Fizyki

Jesień 2013

Semistrukturalny model danych

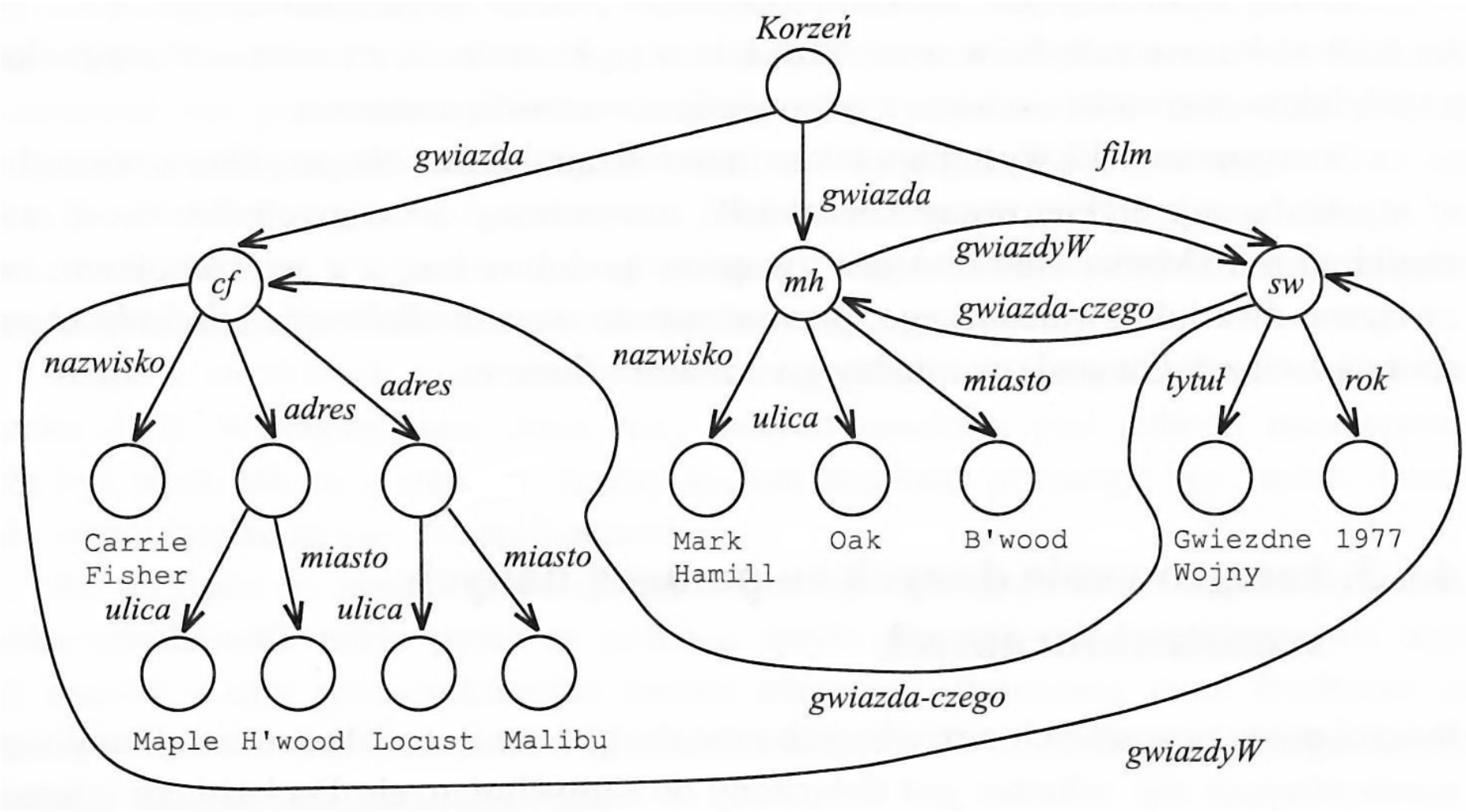
Nie zawsze jest potrzeba albo możliwość posiadania ściśle zdefiniowanego schematu danych.

Semistrukturalny model danych

Nie zawsze jest potrzeba albo możliwość posiadania ściśle zdefiniowanego schematu danych.

W modelu semistrukturalnym dane określają schemat.

Semistrukturalny model danych – przykład



Semistrukturalny model danych c.d.

Zalety

- Swoboda w rozszerzaniu i adaptowaniu do nowych potrzeb.
- Brak ograniczeń wynikających z modelu danych.

Semistrukturalny model danych c.d.

Zalety

- Swoboda w rozszerzaniu i adaptowaniu do nowych potrzeb.
- Brak ograniczeń wynikających z modelu danych.

Wady

- Trudność w formułowaniu zapytań.
- Brak wydajnych implementacji pozwalających na przechowywanie dużych ilości danych.

Semistrukturalny model danych c.d.

Zalety

- Swoboda w rozszerzaniu i adaptowaniu do nowych potrzeb.
- Brak ograniczeń wynikających z modelu danych.

Wady

- Trudność w formułowaniu zapytań.
- Brak wydajnych implementacji pozwalających na przechowywanie dużych ilości danych.

Zastosowania

- Integracja danych pochodzących z różnych źródeł.
- Adaptacja "starych" baz danych do nowych potrzeb.
- Opis dokumentów.

XML

eXtensible Markup Language

Znaczniki

- określają znaczenie podciągów znaków w dokumencie
- teksty ujęte w nawiasy kątowe <...>
- występują w parach – otwierający <...> i zamykający </...>

Zastosowania

- Przechowywanie ustrukturyzowanych danych w plikach tekstowych
- Wymiana danych pomiędzy aplikacjami

Tryby dokumentów XML

Dobrze sformowany XML

- Podejście semistrukturalne
- Dowolne znaczniki
- Brak ustalonego schematu

Ustalony typ dokumentu

- Podejście pośrednie pomiędzy schematem semistrukturalnym, a ścisłymi (np. relacyjnym)
- **D**ocument **T**ype **D**efinition
- Specyfikacja dopuszczalnych znaczników
- Gramatyka zagnieżdżania

Dobrze sformowany XML

Przykład

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<etaty>
    <etat>
        <nazwa>Profesor</nazwa>
        <placa_od>3000</placa_od>
        <placa_do>6000</placa_do>
    </etat>
    <etat>
        ...
    </etat>
</etaty>
```

Dokument ustalony

Dokument bez specyfikacji typu

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
```

Nagłówek

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE typDokumentu SYSTEM "specyfikacja.dtd">
```

Dokument ustalony c.d.

Przykład

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Etaty SYSTEM "etaty.dtd">
<etaty>
    <etat>
        <nazwa>Profesor</nazwa>
        <placa_od>3000</placa_od>
        <placa_do>6000</placa_do>
    </etat>
    <etat>
        ...
    </etat>
</etaty>
```

Dokument ustalony c.d.

- Deklaracja elementu głównego (korzenia)
- Deklaracje elementów
- Deklaracje reguł zagnieżdżania

etaty.dtd

```
<!DOCTYPE etaty [
    <!ELEMENT etaty (etat*)>
    <!ELEMENT etat (nazwa, placa_od, placa_do)>
    <!ELEMENT nazwa (#PCDATA)>
    <!ELEMENT placa_od (#PCDATA)>
    <!ELEMENT placa_do (#PCDATA)>
]>
```

Dokument ustalony c.d.

- Nazwa typu dokumentu: `<!DOCTYPE [...]>`
- Nazwy dopuszczalnych elementów `<!ELEMENT (...)>`
- Reguły zagnieżdżania dopuszczalnych składowych elementów
- element (podelement1*, podelement2+, podelement3?,...)
- Operatory:
  - *: 0 lub więcej
  - +: 1 lub więcej
  - ?: co najwyżej raz
- element (`#PCDATA`)
- Typ `#PCDATA` oznacza dowolny tekst
- Nie występują typy elementów

Przykład

Model relacyjny

Gwiazdy(nazwisko, adres)
Filmy(tytul, rok, dlugosc)
GwiazdyW(tytul, rok, nazwiskoGwiazdy)

Dokument DTD

```
<!DOCTYPE Gwiazdy [
    <!ELEMENT gwiazdy (gwiazda*)>
    <!ELEMENT gwiazda (nazwisko, adres+, filmy)>
    <!ELEMENT nazwisko (#PCDATA)>
    <!ELEMENT adres (\#PCDATA)>
    <!ELEMENT filmy (film*)>
    <!ELEMENT film (tytul, rok, dlugosc)>
    <!ELEMENT tytul (\#PCDATA)>
    <!ELEMENT rok (\#PCDATA)>
    <!ELEMENT dlugosc (#PCDATA)>
]>
```

Przykładowe dane

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy SYSTEM "gwiazdy.dtd">
<gwiazdy>
    <gwiazda>
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>123 Maple St.</adres>
        <filmy>
            <film>
                <tytul>Gwiezdne Wojny</tytul>
                <rok>1977</rok>
                <dlugosc>93</dlugosc>
            </film>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
            <film>
                <tytul>Powrot Jedi</tytul>
                <rok>1983</rok>
                <dlugosc>94</dlugosc>
```

Przykładowe dane

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy SYSTEM "gwiazdy.dtd">
<gwiazdy>
    <gwiazda>
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>123 Maple St.</adres>
        <filmy>
            <film>
                <tytul>Gwiezdne Wojny</tytul>
                <rok>1977</rok>
                <dlugosc>93</dlugosc>
            </film>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
            <film>
                <tytul>Powrot Jedi</tytul>
                <rok>1983</rok>
                <dlugosc>94</dlugosc>
            </film>
        </filmy>
    </gwiazda>
    <gwiazda>
        <nazwisko>Mark Hamill</nazwisko>
        <adres>456 Oak Rd.</adres>
        <adres>789 Pine Av.</adres>
        <filmy>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
```

Dokument ustalony c.d.

- Zagnieżdżanie znaczników nie pozwala przedstawić wszystkich informacji
- Atrybuty pozwalają na dodatkowy opis danych
- Zmniejszenie redundancji
- Mogą służyć do powiązania pojedynczej wartości ze znacznikiem
- Alternatywa dla podznaczników, które są zwykłymi tekstami (`#PCDATA`)

Składnia

- Po deklaracji elementu `<!ELEMENT (...)>`
- `<!ATTLIST element ...>`
- lista atrybutów
  - atrybut1
  - atrybut2 `ID`
  - atrybut3 `IDREF` lub `IDREFS`

Atrybuty – przykład

Dokument DTD

```
<!DOCTYPE Gwiazdy-Filmy [
    <!ELEMENT gwiazdy-filmy (gwiazda*, film*)>
    <!ELEMENT gwiazda (nazwisko, adres+)>
        <!ATTLIST gwiazda
        gwiazdaId ID
        wystepujeW IDREFS>
    <!ELEMENT nazwisko (\#PCDATA)>
    <!ELEMENT adres (ulica, miasto)>
    <!ELEMENT ulica (\#PCDATA)>
    <!ELEMENT miasto (\#PCDATA)>
    <!ELEMENT film (tytul, rok, dlugosc)>
        <!ATTLIST film
        filmId ID
        gwiazdyW IDREFS>
    <!ELEMENT tytul (\#PCDATA)>
    <!ELEMENT rok (\#PCDATA)>
    <!ELEMENT dlugosc (\#PCDATA)>
]>
```

Atrybuty – przykład

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy-Filmy SYSTEM "gwiazdy-filmy.dtd">
<gwiazdy-filmy>
    <gwiazda gwiazdaId="cf" wystepujeW="gw, ik, pj">
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>
            <ulica>123 Maple St.</ulica>
            <miasto>Hollywood</miasto>
        </adres>
    </gwiazda>
    <gwiazda gwiazdaId="mh" wystepujeW="ik, pj">
        <nazwisko>Mark Hamill</nazwa>
        <adres>
            <ulica>456 Oak Rd.</ulica>
            <miasto>Malibu</miasto>
        </adres>
        <adres>
            <ulica>123 Pine Av.</ulica>
            <miasto>Brentwood</miasto>
        </adres>
    </gwiazda>
```

Atrybuty – przykład

```
    <film  filmId="gw"  gwiazdyW="cf">
        <tytul>Gwiezdne Wojny</tytul>
        <rok>1977</rok>
        <dlugosc>93</dlugosc>
    </film>
    <film  filmId="ik"  gwiazdyW="cf, mh">
        <tytul>Gwiezdne Wojny</tytul>
        <rok>1980</rok>
        <dlugosc>96</dlugosc>
    </film>
    <film  filmId="pj"  gwiazdyW="cf, mh">
        <tytul>Powrot Jedi</tytul>
        <rok>1983</rok>
        <dlugosc>94</dlugosc>
    </film>
</gwiazdy-filmy>
```

Document Object Model – DOM

Konwencja opisu dokumentu XML (HTML, XHTML) pozwalająca na dostęp do poszczególnych elementów oraz wprowadzanie zmian w dokumencie.

Implementacje w przeglądarkach webowych (JavaScript) oraz bibliotekach obsługujących pliki XML.

W Pythonie moduł `xml.dom`.

`xml.dom` – podstawowe API

Klasy

Document	Cały dokument
Node	Klasa, z której dziedziczą składniki dokumentu
Element	Elementy XML
Attr	Atrybuty elementów
Comment	Komentarze
Text	Napisy (tekstowa zawartość elementów)

xml.dom.minidom – minimalistyczna implementacja DOM

Funkcje

xml.dom.minidom.parse(filename_or_file) Wczytuje dokument z pliku lub obiektu plikowego

xml.dom.minidom.parseString(string) Wczytuje dokument z napisu

Przykład

```
>>> d=xml.dom.minidom.parseString("<node>Text</node>")
>>> d
<xml.dom.minidom.Document instance at 0x105bed638>
>>> d.documentElement
<DOM Element: node at 0x105bed680>
>>> d.documentElement.tagName
u'node'
>>> d.documentElement.childNodes
[<DOM Text node "u'Text'">]
```

Modyfikacje DOM

```
>>> newel=d.createElement('extranode')
>>> nodetext=d.documentElement.childNodes[0]
>>> d.documentElement.insertBefore(newel, nodetext)
<DOM Element: extranode at 0x105bed3f8>
>>> newel=d.createElement('extranode')
>>> d.documentElement.appendChild(newel)
<DOM Element: extranode at 0x105ac7320>
>>> d.documentElement.childNodes
[<DOM Element: extranode at 0x105bed3f8>, <DOM Text node "u'Text'">, <DOM Element:
>>> d.documentElement.childNodes[0].appendChild(d.createTextNode('Extra1'))
<DOM Text node "'Extra1'">
>>> d.documentElement.childNodes[2].appendChild(d.createTextNode('Extra2'))
<DOM Text node "'Extra2'">
>>> d.documentElement.toxml()
u'<node><extranode>Extra1</extranode>Text<extranode>Extra2</extranode></node>'
>>> d.documentElement.removeChild(nodetext)
<DOM Text node "u'Text'">
>>> d.documentElement.toxml()
u'<node><extranode>Extra1</extranode><extranode>Extra2</extranode></node>'
>>>
```

Wyszukiwanie

tree.xml

```
<!DOCTYPE tree [<!ATTLIST node id ID #IMPLIED>]>
<tree>
    <node type="domain" id="Bacteria"></node>
    <node type="domain" id="Archaea"></node>
    <node type="domain" id="Eukaryota">
        <node type="supergroup" id="Archaeplastida"></node>
        <node type="supergroup" id="Opisthokonta">
            <node type="kingdom" id="Fungi">
                <node type="genus" id="Muchomorek"></node>
            </node>
            <node type="kingdom" id="Animalia">
                <node type="type" id="Chordata">
                    <node type="class" id="Amphibia">
                        <node type="genus" id="Zielona␣zabka"></node>
                    </node>
                    <node type="class" id="Sauropsida"></node>
                    <node type="class" id="Aves"></node>
                    <node type="class" id="Synapsida"></node>
                    <node type="class" id="Mammalia">
                        <node type="genus" id="Myszka␣mala"></node>
                    </node>
                </node>
                <node type="type" id="Arthropoda"></node>
```

Wyszukiwanie c.d.

```
>>> d=xml.dom.minidom.parse('tree.xml')
>>> myszka=d.getElementById('Myszka mala')
>>> myszka
<DOM Element: node at 0x105be4a28>
>>> x=myszka
>>> while x.tagName=='node':
...     res.append(x)
...     x=x.parentNode
...
>>> res
[<DOM Element: node at 0x105be4a28>, <DOM Element: node at 0x105bed128>,
<DOM Element: node at 0x105bf1248>, <DOM Element: node at 0x105bf0f80>,
<DOM Element: node at 0x105bf07a0>, <DOM Element: node at 0x105bf0290>]
>>> [(x.getAttribute('type'), x.getAttribute('id')) for x in res]
[(u'genus', u'Myszka mala'), (u'class', u'Mammalia'), (u'type', u'Chordata'),
(u'kingdom', u'Animalia'), (u'supergroup', u'Opisthokonta'),
(u'domain', u'Eukaryota')]
>>>
```

gnosis.xml

http://www.gnosis.cx/download/Gnosis_Utils.More/Gnosis_Utils-1.2.2.tar.gz

Moduł służący do konwersji XML na hierarchie zwykłych obiektów.

gnosis.xml c.d.

Przykład

```
>>> d=objectify.make_instance("tree1.xml")
>>> d
<tree id="107256890">
>>> d.__dict__
'_seq': [u'\n    ', <node id="1072568d0">, u'\n    ', <node id="107256850">,
u'\n    ', <node id="107256950">, u'\n'], 'node': [<node id="1072568d0">,
<node id="107256850">, <node id="107256950">], 'PCDATA': '', '__parent__': None
>>> d.node
[<node id="1072568d0">, <node id="107256850">, <node id="107256950">]
>>> d.node[0].__dict__
u'type': u'domain', u'id': u'Bacteria', '__parent__': <tree id="107256890">
>>> [x.id for x in d.node]
[u'Bacteria', u'Archaea', u'Eukaryota']
```

gnosis.xml c.d.

Wypisywanie

```
>>> objectify.utils.write_xml(d)
<tree>
    <node type=domain id=Bacteria></node>
    <node type=domain id=Archaea></node>
    <node type=domain id=Eukaryota>
        <node type=supergroup id=Archaeplastida></node>
        <node type=supergroup id=Opisthokonta>
            <node type=kingdom id=Fungi>
                <node type=genus id=Muchomorek></node>
            </node>
            <node type=kingdom id=Animalia>

...

            </node>
        </node>
    </node>
</tree>>>>
>>> d.label="Tree of life"
>>> objectify.utils.write_xml(d)
<tree label=Tree of life>
    <node type=domain id=Bacteria></node>
    ...
```

gnosis.xml c.d.

Obchodzenie drzewa

```
>>> objectify.utils.walk_xo(d)
<generator object walk_xo at 0x107315b90>
>>> list(objectify.utils.walk_xo(d))
[<tree id="107256890">, <node id="1072568d0">, <node id="107256850">,
<node id="107256950">, <node id="1072569d0">, <node id="107256b10">,
<node id="107256b50">, <node id="107256b90">, <node id="107256bd0">,
<node id="107256c10">, <node id="107256c50">, <node id="107256c90">,
<node id="107256cd0">, <node id="107256d10">, <node id="107256d50">,
<node id="107256d90">, <node id="107256dd0">, <node id="107256e10">]
>>>
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

```
>>> def node_repr(self):
...     return "'"+self.type+": "+self.id+"'"
...
>>> gnosis.xml.objectify._XO_node.__repr__=node_repr
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

```
>>> def node_repr(self):
...     return "'"+self.type+": "+self.id+"'"
...
>>> gnosis.xml.objectify._XO_node.__repr__=node_repr
```

W00t

```
>>> list(objectify.utils.walk_xo(d))
[<tree id="107256890">, 'domain: Bacteria', 'domain: Archaea',
'domain: Eukaryota', 'supergroup: Archaeplastida', 'supergroup: Opisthokonta',
'kingdom: Fungi', 'genus: Muchomorek', 'kingdom: Animalia', 'type: Chordata',
'class: Amphibia', 'genus: Zielona zabka', 'class: Sauropsida', 'class: Aves',
'class: Synapsida', 'class: Mammalia', 'genus: Myszka mala', 'type: Arthropoda']
```

Czym są elementy dokumentu w gnosis.xml?

Można *a priori* zdefiniować klasy odpowiadające elementom.

```
class Node(gnosis.xml.objectify._XO_):
    def repr(self):
        return "'"+self.type+": "+self.id+"'"

gnosis.xml.objectify._XO_node=Node
```

Zadanie 1,2 – Edycja dokumentu

Zadanie

Dopisz kilka ulubionych taksonów do `tree.xml`. Wykonaj to przy pomocy `xml.dom.minidom` i `gnosis.xml`.

Zadanie 3 – Bakterie

Zadanie

Opisz format dokumentu XML przechowującego stan szalki Petriego z poprzednich zajęć (informacje o zdarzeniach, które mają zajść pomijamy). Zrealizuj możliwość startu symulacji od wczytanego stanu.