PROGRAMAREA ORIENTATĂ PE OBIECTE
ȘI
LIMBAJELE SEMANTICE

IORDACHE RAUL-MIHAI
406, INGINERIE SOFTWARE

CUPRINS

1. Introducere. Motivație

- Concept relativ nou: 2009 (SOOP, 2016)
- OOP = o paradigmă foarte puternică în industria software
- Semantica = o necesitate, pentru a putea manipula informații eterogene
- OOP nu este potrivită pt. prelucrarea și procesarea datelor semantice

1. Introducere. Motivație

- Semantica (din Semantic Web) este mult mai puternică decât ,,semantica” modelată în context OOP
- Motivație: combinarea a două concepte auxiliare, pentru rezolvarea problemelor complexe (care utilizeaza date variate, din surse diferite)

2. Semantic Oriented Framework

- Design-ul object-oriented + elemente de Semantic Web = SOF
- Utilizeaza comentarii înglobate în codul sursă pentru a descrie relații semantice (între clase și/sau atribute)

2. Semantic Oriented Framework

Problemă	Jena	Active RDF	D2R	Eclass	*SOF*
Utilizarea metodelor pentru a manipula RDF	Nu	Da	Nu	Da	Da
Conversie automată în format RDF	Nu	Nu	Da	Da	Da
Suport pentru mai multe limbaje de progr.	Nu	Nu	Da	Nu	Da
Utilizarea directivelor pentru descrierea claselor și a semanticii atributelor	Nu	Nu	Nu	Da	Da
Actualizarea fișierelor de descriere semantică	Nu	Nu	Nu	Da	Da
Suport pentru surse de date heterogene	Nu	Nu	Nu	Nu	Da
Verificarea consistenței datelor (și a semanticii)	Nu	Nu	Nu	Nu	Da

2. Semantic Oriented Framework

- **Module componente a SOF:**
  - SOF data adaptor:
    - Citește sursa (CSV, înregistrări DB), output: obiecte (model obj)
  - SOF parser:
    - Citește cod OO (comentariile din sursă), output: obiecte ale ontologiei
  - SOF RDF generator:
    - Input: model obj + onto obj; Output: Model obj în format RDF (file)
  - SOF Query Engine:
    - Input: interogări semantice; Output: Obiecte (răspunsul)

2. Semantic Oriented Framework

- **Exemplu:**

```
typedef class Angajat {
string nume;
// owl: InverseFunctionalProperty Angajat_email
string email;
Telefon NrTel;
// Angajat_NrTelServiciu rdfs:subClassOf Angajat_NrTel;
Telefon NrTelServiciu;
// Angajat_NrTelPersonal rdfs:subClassOf Angajat_NrTel;
Telefon NrTelPersonal;
//....
};
```

A se observa semantica adăugată.

3. Semantic Object-Oriented Programming

- Crearea și popularea ontologiei direct în C++ .
- Se utilizează clase, obiecte, șabloane (template) exclusive C++ (în mod general, OOP).

OOP	Vocabular comun	Ontologii
Clasă Obiect (instanță)	Entitate	Atom, Axiomă, Variabilă (cu sens diferit față de OOP), Predicat, Cuantor

3. Semantic Object-Oriented Programming

Ontologie → OOP	OOP → Ontologie
Maparea relațiilor dinamice a entităților într-un tip static - **imposibilă**	Maparea tipului relației (al unui tip static) într-o ontologie dinamica – **posibilă (f. ușor)**
Obiecte arbitrare pot fi legate utilizând relații arbitrare. Deci, **translatare grea** (în atribut obiectual)	Conexiunile atributelor sunt destul de simple în OOP, și **ușor de mapat** într-o ontologie
O ontologie este un hipergraf care prezintă (în general) un set complex de relații. Foarte puțin probabilă **maparea** (chiar dacă limbajul suportă moștenirea multiplă)	Moștenirea directă, aciclică din OOP este defapt un subgraf (al unei ontologii mari – hipergraf -)
Mecanismul ontologiei este tocmai evoluția dinamică a structurii. Un lucru destul de greu de translatat într-un limbaj static.	Structurile moștenite (din OOP) sunt foarte ușor de transformat într-o ontologie dinamică

3. Semantic Object-Oriented Programming

- Exemplu:

```cpp
#include <iostream>
#include <string>
#include <soop/onto.hpp>

class angajat {
public:
    angajat(std::string nume) : nume{ std::move(nume) } {}
    std::string nume;
};

using angajat_e = soop::e<angajat>;
SOOP_MAKE_TYPECHECKED_PREDICATE(parent_of, 2, angajat_e, angajat_e);
//predicatul parent_of cu sensul relatiei: sef-angajat

int main() {
using namespace preds;
using namespace soop::preds;
soop::ontology o{};
o.add_type<angajat_e>();
o.add_predicate<parent_of_t>();
soop::variable<'sef'> sef;
soop::variable<'subaltern'> subaltern;

// seful si subalternul sunt angajati:
o.add_axiom(forall({ sef, subaltern }, implies(parent_of(sef, subaltern), and_(
        instance_of(sef, soop::type<angajat_e>),
        instance_of(subaltern, soop::type<angajat_e>)))));

// axioma: seful unui subaltern, nu este subalternul unui sef:
    o.add_axiom(forall({ sef, subaltern }, implies(parent_of(sef, subaltern),
        not_(parent_of(subaltern, sef)))));
    angajat_e PopescuSef{ o, "Popescu Alexandru" };
    angajat_e IonescuSubaltern{ o, "Ionescu Adrian" };
    o.add_axiom(preds::parent_of(PopescuSef, IonescuSubaltern));
    soop::variable<'s'> s;
    const auto& sef = std::get<0>(
        o.request_entities<angajat_e>(
            exists({ subaltern }, parent_of(s, subaltern)), s));
    std::cout << sef->nume << " este un sef.\n";
}
```

4. SOF vs. SOOP

SOF	SOOP
- adăugarea semanticii sub formă de comentarii - nu se alterează codul original (semantica poate fi adăugată ulterior) - mapare indirectă semantică<->OOP - tehnică mai complexă (necesită un parser, etc.) și mai matură - necesită cunoașterea limbajelor de modelare semantică	- integrarea (modelarea) în cod a ideilor semantice - se modifică structura codului sursă, prin unele modificări (utilizarea unor template-uri din cadrul SOOP, etc.) - creare directă a ontologiei în OOP - tehnică mai tânără - nu necesită cunoașterea limbajelor semantice. Totuși necesită cunoștințe solide de modelare semantică .... (+ logică declarativă + programare logică (Prolog e un plus))

5. Concluzii

- Fiecare tehnologie are avantajele și dezavantajele ei.
- În funcție de cerințele problemei + experiența programatorului în materie de S.W., utilizăm una dintre abordări.
- În principiu, dacă suntem mai siguri pe experiența OOP, alegem SOOP, în caz contrar alegem SOF.

Vă mulțumesc pentru atenție!