CONFÉRENCE UNIVERSITAIRE
DE SUISSE OCCIDENTALE

UNIVERSITÉ DE
NEUCHÂTEL

On The New Prospects for Knowledge Representation in The Time of Dynamic Compilation

Selena Baset

Information Management Institute - Faculty of Economics and Business

Thesis supervisor:

Prof. Kilian Stoffel

Problem Statement

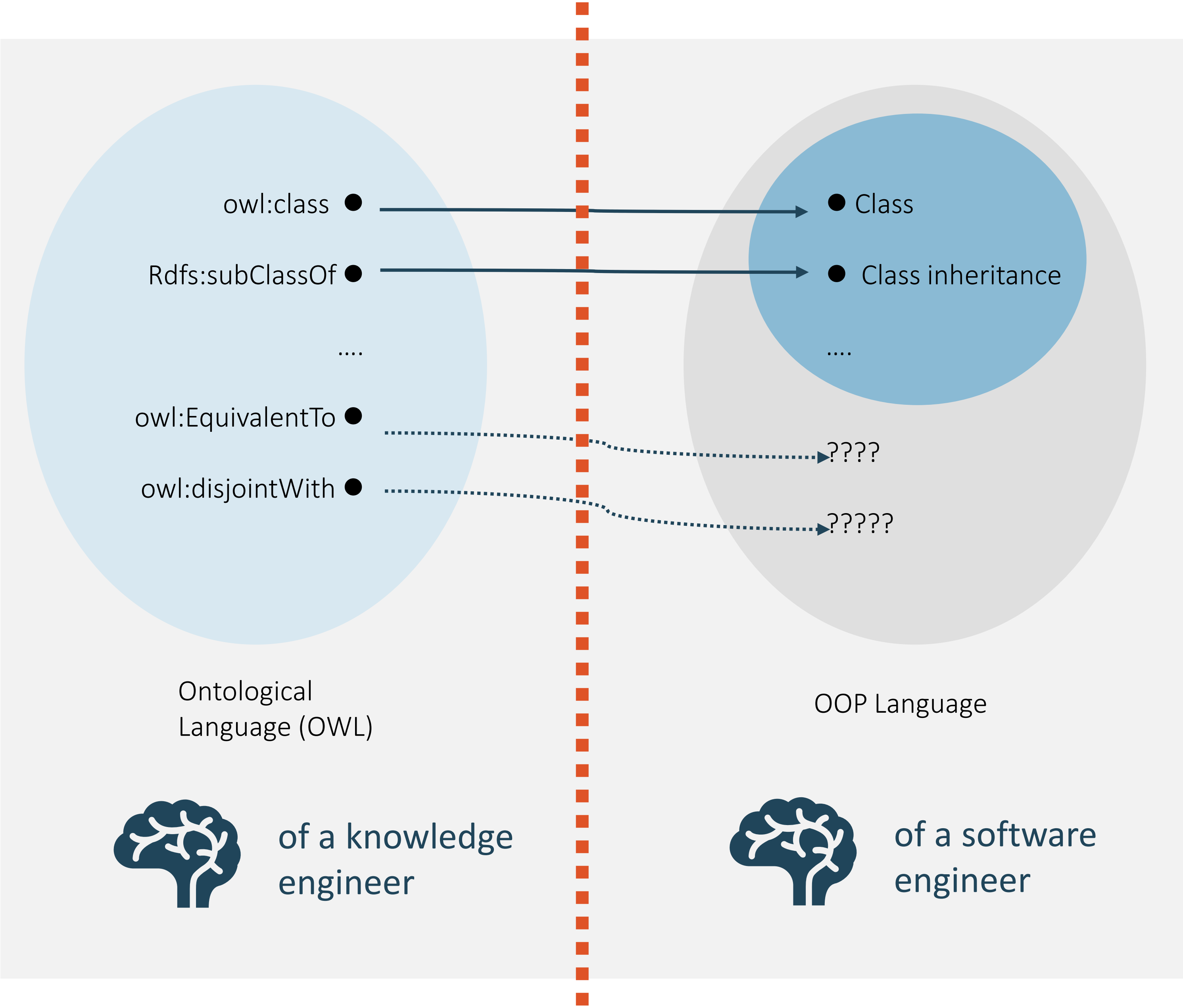
Most efforts on understanding, incorporating and using ontologies in mainstream software development are hindered by a <u>fundamental issue</u>:

The semantic gap between two schools of modeling; the school of KR&R and that of Software Engineering.

- Ontological languages are declarative and way more expressive than formal (imperative) programming languages.
- The different assumptions on which reasoning in these two schools is based on: OWA vs CWA.

The problem in other words

Finding a total transformation function that pairs every syntactic construct from the ontological domain with its corresponding OOP construct while preserving the semantics is not straightforward.



Software Engineering:

- Along the process of software development, it is highly desirable to have software models systematically connected and used collaboratively, rather than in isolation.
- With the lack of technical tools to support and facilitate the integration, many software developers find ontologies more of hurdle than help.

Knowledge Engineering

Executable ontologies can benefit from the new palette of metaprogramming and dynamic compilation tools offered by the language compiler. Such features can be exploited to add procedural extensions or to explore new possibilities of entailment reasoning.

Relevancy

Research questions

- Would expressing ontologies in a mainstream programming language (or paradigm) help in lessening the difficulties in integrating them into a conventional codebase?
- By which means could we achieve this potential translation into executables without impacting the expressiveness of the ontology?
- If the domain knowledge is automatically translated into executables, how would this affect development time?
- What are the potential new forms of entailment in such settings? To which extent can we exploit the language compiler’s features in supporting inference tasks?

H_0 Hypotheses

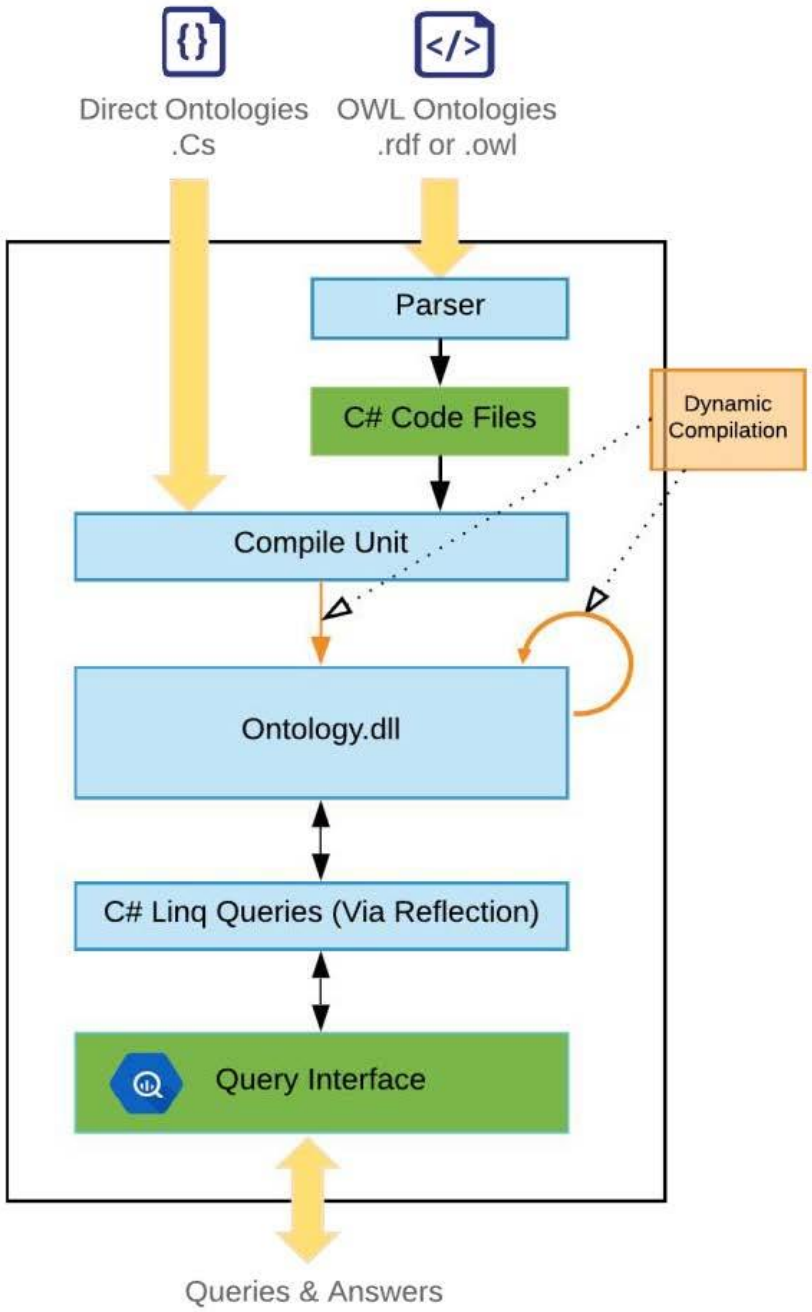
- Expressing ontologies in a mainstream programming language can bring them into the comfort-zone of software developers and thus in integrating KR concepts into conventional codebases.
- Metaprogramming and dynamic compilation can support lossless transformation of ontologies into executables in the target programming language.
- Having domain knowledge readily translated into executables results in shorter episodes of code refactoring in response to changing requirements.
- The compiler built-in support for object-oriented inheritance can be exploited to readily infer some of the implicit relations in the concepts' hierarchy.

Related work

- Work on Model Driven Development MDD (Atkinson, 2003) and Ontology Driven Software Development ODSD (Knublauch, 2004).
- Enterprise Java Beans and the development of semantic-rich enterprise applications (Athanasiadis et al., 2007).
- Hybrid modeling software framework: object-oriented & ontological representation; advantages and disadvantages (Puleston, 2008).
- OWL to UML mapping (Atkinson, 2006) and (Brockmans et al., 2004).
- Recent work on Ontology-oriented Programming in Python (Lamy,2017)

Approach

- Feasibility study and proof by construction.
- A prototype for an ontological knowledgebase system where ontologies are expressed in an executable form.
- OOP as programming paradigm and C# as both representation and query language.

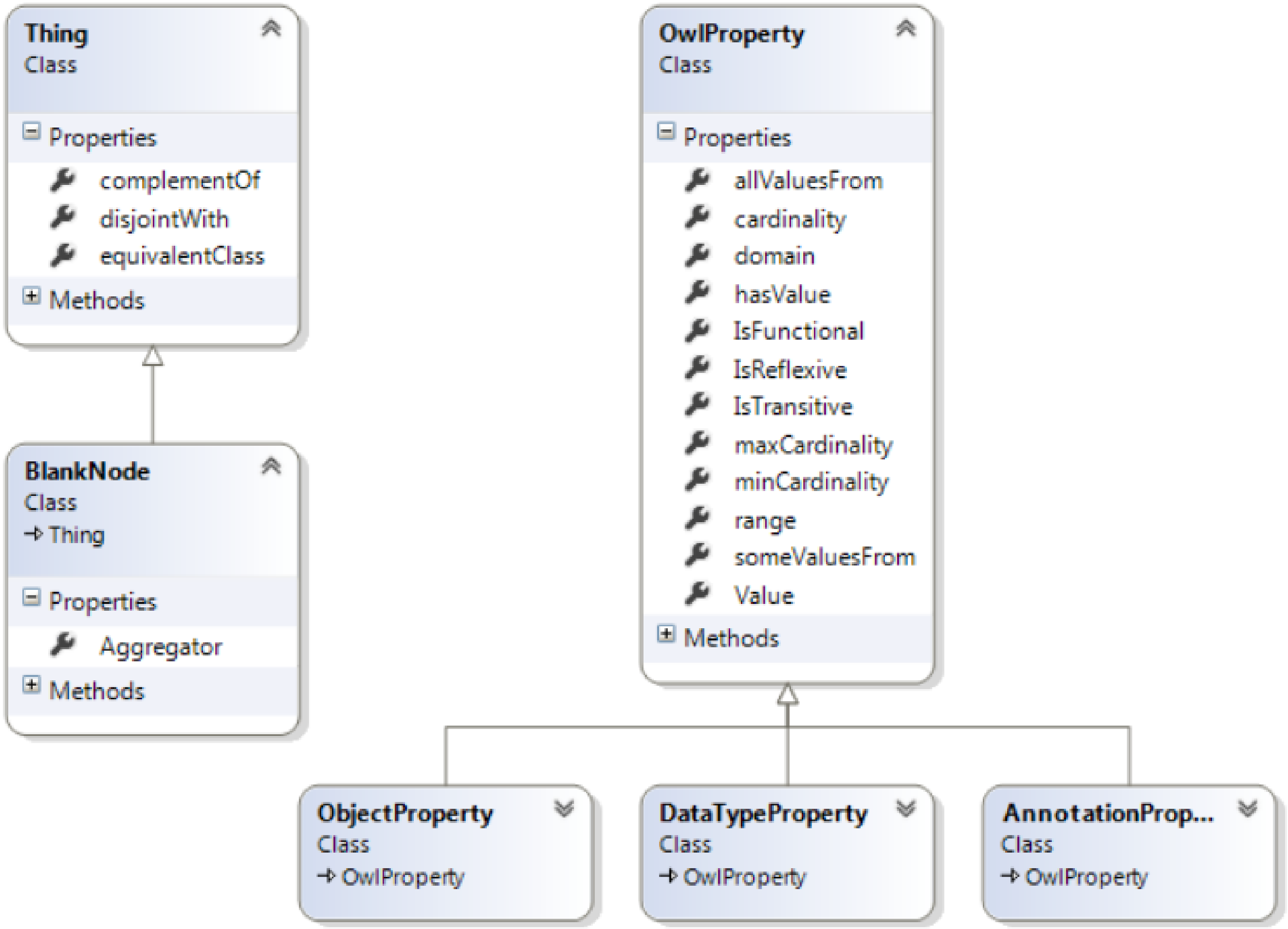


Architecture

OWL to OOP Mapping

Axiom	OWL	OntoJIT Counterpart
Ontology	owl:Ontology	Code namespace
Class	owl:class	C# class
	rdfs:subclass	C# class inheritance
Class Description	rdfs:equivalentClass	Static meta properties
	owl:intersectionOf	
	owl:unionOf	
	owl:complementOf	
	owl:disjointWith	
Indivisual	indivisual	object instance
	owl:AllDifferent	Non-static meta properties
	owl:differentFrom	
	owl:sameAs	
Property	owl:ObjectProperty	C# class
	owl:DataTypeProperty	
	rdfs:subPropertyOf	C# class inheritance
Property Association	rdfs:range	Static meta properties
	rdfs:domain	
Property Restriction	rdfs:cardinality	Static meta properties
	rdfs:hasValue	
	rdfs:someValuesFrom	
	rdfs:allValuesFrom	
Property Description	owl:FunctionalProperty	Static meta properties
	owl:InverseFunctionalProperty	
	owl:SymmetricProperty	
	owl:TransitiveProperty	
Property Relations	owl:inverseOf	Static meta properties
	owl:subPropertyOf	
	owl:equivalentProperty	

Initial heirarchy



Generated Code

```
public class American : NamedPizza
{

    public static hasTopping hasTopping;
    public static object subClassOf;
    public static hasCountryOfOrigin hasCountryOfOrigin;
    public static string label;

    static American()
    {
        label = "en:American ; pt:Americana";
        hasCountryOfOrigin = new hasCountryOfOrigin();
        subClassOf = MozzarellaTopping;
        hasTopping.someValuesFrom =
            new List<PizzaTopping>()
                { MozzarellaTopping,
                  PeperoniSausageTopping,
                  TomatoTopping
                };
        hasTopping.allValuesFrom = MozzarellaTopping;
        hasCountryOfOrigin.hasValue = America;
        hasTopping = new hasTopping();
    }

}
```

```
public class GO_0048311 : GO_0007005
{
    public static string label;
    public static object subClassOf;

    static GO_0048311()
    {
        subClassOf = GO_0051646;
        label = "mitochondrion distribution";
    }
}

public class GO_0000002 : GO_0007005
{
    public static string label;

    static GO_0000002()
    {
        label = "mitochondrial genome maintenance";
    }
}

public class GO_0007005 : GO_0006996
{
    public static string label;

    static GO_0007005()
    {
        label = "mitochondrion organization";
    }
}
```

Query Example

Given the gene ontology, find all chromosomes that are part of a cytoplasm.

In Description Logic:

$$\exists x.(chromosome(x) \wedge is\_part(x,y) \wedge cytoplasm(y))$$

In C#:

```
var chromosomes = TransitiveClosure(typeof(chromosome));
var cytoplams = TransitiveClosure(typeof(cytoplasm));

var types =
    from type
    in Assembly.GetExecutingAssembly().GetTypes()
    let property = type.GetField("is_part")
    let value = (ObjectProperty)property.GetValue(null)
    where chromosomes.Contains(type) &&
    cytoplams.Intersect(UnwrapValue(value.someValuesFrom)).Count() > 0
    select type;
```

GO_0000229: cytoplasmic chromosome
GO_0000262: mitochondrial chromosome
GO_0009508: plastid chromosome
GO_0042648: chloroplast chromosome

Validation Plan

- The proof by construction approach and the proposed prototype can only validate some of the hypotheses.

- Impact of development life-cycle to be assessed properly in the context of medium to long term projects using surveys, focus groups discussions and agile process metrics (Leadtime, Cycle time, team velocity, etc).

Preliminary results

Representation:

- We have tested the parsing component against ontologies of varying size and expressivity profiles: Gene, wine, pizza ontologies and others.
- Lossless translation for OWL Lite and OWL DL profiles but not for OWL FULL.

Reasoning:

- When testing C# Linq queries that are equivalent to state of the art DL queries, test results were identical except some cases involving OWA reasoning.

Reflections and Outlook

Expressing domain knowledge as executable-ontologies in an OOP language might be an idea in its infancy stage but it has many potentials in the two universes of software development and knowledge representations alike.

But…

- It is hard to validate methodological hypotheses.
- One needs really to **mind the** ‘semantic’ **gap**.

Thank you!