IMPLEMENTING BETTER ONTOLOGIES WITH GUFO
A HANDS-ON TUTORIAL

Tiago Prince Sales, João Paulo A. Almeida, Giancarlo Guizzardi

t.princsales@utwente.nl
TEAM AND ACKNOWLEDGEMENTS

Tiago Prince Sales  
University of Twente,  
The Netherlands

Giancarlo Guizzardi  
University of Twente,  
The Netherlands

João Paulo A. Almeida  
Federal University of Espírito Santo, Brazil

Claudenir M. Fonseca  
University of Twente,  
The Netherlands

... and all who contributed to UFO over the years!
TARGET AUDIENCE AND GOAL

• Target audience
  • Researchers and practitioners interested in designing better OWL ontologies

• Requirements
  • You know how to build ontologies in OWL using a tool like Protégé
  • No previous knowledge of UFO or OntoUML is required

• Learning objectives
  • Knowledge on how to use gUFO to create an ontology in OWL
  • Knowledge on how to apply gUFO's patterns to solve recurrent modeling problems
AGENDA

• Part 1
  • Introduction
  • Getting started with gUFO

• Part 2
  • Taxonomy of individuals and object properties
  • Qualities and datatype

• Part 3
  • Taxonomy of types
  • Historical data
  • Closing
APPREACH

REFERENCE ONTOLOGY X ONTOLOGY IMPLEMENTATION

• Reference ontology
  • Is built as a conceptual model giving precedence to real-world adequacy
  • Designed for a class of problems
  • UFO is a reference ontology

• Ontology Implementation
  • Sacrifices real-world adequacy to obtain computational properties
  • Designed for a specific problem
  • gUFO is our implementation of UFO in OWL
    • ‘g’ stands for gentle
APPROACH

Domain-independent

More specific

Common Ontology of Value and Risk

Your core ontology here!

AlpineBits DestinationData Ontology

Your domain ontology here!

gUFO
“We shall do a much better programming job, provided that we approach the task with a full appreciation of its tremendous difficulty, […] we stick to modest and elegant programming languages, […] we respect the intrinsic limitations of the human mind and approach the task as Very Humble Programmers.”

IMPLEMENTING BETTER ONTOLOGIES

- We need all the help we can get!
- Reuse of definitions and rules in foundational layer
  - “a little semantics goes a long way” – James Handler
  - “some more semantics goes further” – João Paulo A. Almeida
- Patterns all the way
  - cope with recurrent conceptual challenges
  - cope with recurrent implementation challenges
  - improve implementation stability
- Automatic error detection
  - beyond what can be achieve in the ontologically-neutral OWL
trophic (Q381165)

reward for a specific achievement

Trophy Cup

In more languages

Statements

<table>
<thead>
<tr>
<th>subclass of</th>
<th>award</th>
<th>edit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 references</td>
<td>+ add reference</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prize</th>
<th>edit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 references</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>work of art</th>
<th>edit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 references</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>physical object</th>
<th>edit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 references</td>
</tr>
</tbody>
</table>

+ add value
award (Q618779)

something given to a person or a group of people to recognize their merit or excellence

medal | honour | honor | prize | Awards and Prizes

› in more languages

Statements

subclass of

recognition

○ 0 references

grant

○ 0 references

image

Medalsofhonor2.jpg
675 x 340; 71 KB

○ 0 references
recognition (Q7302601)

public acknowledgement of person's status or merits

› In more languages

Statements

<table>
<thead>
<tr>
<th>subclass of</th>
<th>social status</th>
<th>thanking</th>
<th>acknowledgement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 references</td>
<td>0 references</td>
</tr>
</tbody>
</table>

Identifiers

<table>
<thead>
<tr>
<th>GND ID</th>
<th>4128520-7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 reference</td>
</tr>
</tbody>
</table>

| Art & Architecture Thesaurus ID | 300225689 | 0 references |
thanking (Q83493482)

expression of gratitude for an action
thank you! thanks

• in more languages

Statements

instance of
• activity
• 0 references

subclass of
• acknowledgement
• 0 references

• occurrence
• 0 references

part of
• politeness
• 0 references

has cause
• gratitude
• 0 references
occurrence (Q1190554)

occurrence of a fact or object in space-time; instantiation of a property in an object
occurant | perdurant | event | occurrences | occurants | perdurants | incident | occurring

Statements

<table>
<thead>
<tr>
<th>property</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>subclass of</td>
<td>temporal entity</td>
</tr>
<tr>
<td>has cause</td>
<td>cause</td>
</tr>
<tr>
<td>has effect</td>
<td>effect</td>
</tr>
</tbody>
</table>

0 references
OOPS!

But these are disjoint classes!

- Trophy
  - Physical Object
    - Concrete Object
      - Object
  - Award
    - Recognition
      - Occurrence
thanking  (Q83493482)

expression of gratitude for an action
thank you | thanks

Statements

instance of
- activity
- 0 references

subclass of
- acknowledgement
- 0 references

occurrence
- 0 references

part of
- politeness
- 0 references

has cause
- gratitude
- 0 references
activity (Q1914636)

series of actions which results in a change of state
act | action | measure

Statements

<table>
<thead>
<tr>
<th>subclass of</th>
<th>series</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 references</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 references</td>
</tr>
</tbody>
</table>

image

Kamplanzug See.jpg
1,280 × 960; 419 KB

0 references
OOPS AGAIN!

Thanking is both a subclass and an instance of occurrence!

“series of actions which results in a change of state”
FOUNDATIONAL ONTOLOGIES

- What is a foundational ontology?
  - Captures our understanding of general (ubiquitous!) notions
  - Objects, their aspects, their types, their parts, … events, situations…
FOUNDATIONAL ONTOLOGIES

Why should I use a foundational ontology when creating my OWL ontologies?

- You get a “seed ontology” from which you can build your own ontology
- You reuse domain independent concepts
- You avoid conceptual mistakes
- You increase the “semantic depth” of your ontology, improving its interoperability
FOUNDATIONAL ONTOLOGIES

Why should I use gUFO as my foundational ontology in OWL?

- You get to use foundational patterns to model:
  - Roles
  - Qualities
  - Phases
  - Relationships

- You can express that not all types are “the same”

- You get a sophisticated theory of relationships

- You get support for multi-level modeling

- You get patterns to handle change and historical data
GUFO OVERVIEW

- gUFO reflects UFO taxonomies of individuals and types (universals)
- We slightly adjust the terminology (when possible) to avoid philosophical jargon
REUSABLE CLASSES

gUFO

gUFO-based (domain) ontology
REUSABLE PROPERTIES

gUFO-based (domain) ontology

More than a taxonomy!
GETTING STARTED
RESOURCES

1. gUFO: https://purl.org/nemo/gufo
2. gUFO Documentation: https://purl.org/nemo/doc/gufo
3. gUFO YouTube Playlist: https://www.youtube.com/playlist?list=PL4-CtXCqPknOLd3KAr8Oygk0dyFlOajdM
4. gUFO Protégé Plugin (Prototype): https://github.com/nemo-ufes/ufo-protege-plugin
5. gUFO 101: https://github.com/unibz-core/gufo-tutorial-ontobras
The objective of uGuFO is to provide a lightweight implementation of the Unified Foundational Ontology (UFO) suitable for Semantic Web OWL 2 DL applications. Intended users are those implementing UFO-based lightweight ontologies that reuse gUFO by specializing and instantiating its elements.

There are three implications of the use of the term lightweight. First of all, we have employed little expressive means in an effort to retain computational properties for the resulting OWL ontology. Second, we have selected a subset of UFO-A [1, 2] and UFO-B [3] to include here. In particular, there is minimalistic support for UFO-B (only that which is necessary to establish the participation of objects in events and to capture historical dependence between events). Third, a lightweight ontology, differently from a reference ontology, is designed with the purpose of providing an implementation artifact to structure a knowledge base (or knowledge graph). This has driven a number of pragmatic implementation choices which are discussed in comments annotated to the various elements of this implementation.

The 'g' in gUFO stands for gentle. At the same time, "gufo" is the Italian word for "owl".

For background information on the reference ontology on which this implementation is based, see:
gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO)

**IRI**

http://purl.org/nemo/gufo#

**Creator(s)**
- Almeida, João Paulo A.
- Falbo, Ricardo A.
- Guizzardi, Giancarlo
- Sales, Tiago P.

**Version Information**

1.0.0

**License**

https://creativecommons.org/licenses/by/4.0/legalcode

**Ontology Source**

RDF (Turtle)

**Description**

The objective of gUFO is to provide a lightweight implementation of the Unified Foundational Ontology (UFO) [1-5] suitable for Semantic Web OWL 2 DL applications.
gUFO (gentle Unified Foundational Ontology)

5 videos • 149 views • Last updated on 5 Oct 2021
1. Import gufo using https://purl.org/nemo/gufo

Import using HTTPS, not HTTP
1. Import gufo using https://purl.org/nemo/gufo

2. Add the gufo prefix:
   gufo: http://purl.org/nemo/gufo#

Now, use HTTP instead
1. Import gufo using [https://purl.org/nemo/gufo](https://purl.org/nemo/gufo)

2. Add the gufo prefix:


3. If needed, show the imports closure of your ontology
A Protégé plugin for gUFO-based ontologies

This Protégé plugin extends the tool with functionality for gUFO-based ontologies. In particular, it verifies a number of rules that a gUFO-based ontology should satisfy (based on [4]), supporting a user in checking the quality of the ontology implementation. Experimental support for gUFO patterns is also available, with wizards to...
1. Navigate to https://github.com/nemo-ufes/ufo-protege-plugin

2. Download the latest release from the Release section

   ufo-protege-plugin-0.0.9.jar

3. Copy the downloaded file to Protégé’s plugin folder
   1. On macOS, right-click the Protégé app and select “Show Package Contents”
   2. Navigate to “Contents > Java > plugins” and copy the downloaded file there
   3. Restart Protégé

4. On Protégé top menu, go to “Window > Tabs > UFO Validation Tab”
As a kind, Man cannot specialize a sortal (Person).

The sub kind Robocop must specialize only one kind, but it is specializing many (Machine, Person).
There are 4 ways in which you can reuse gUFO classes:

1. By **instantiating** classes in the taxonomy of **individuals**
2. By **specializing** classes in the taxonomy of **individuals**
3. By **instantiating** classes in the taxonomy of **types**
4. By **specializing** classes in the taxonomy of **type**

Users may combine these various approaches.

By default, we recommend employing scenarios 2 and 3 together.
REUSING GUFO CLASSES (1)

1. By **instantiating** classes in the taxonomy of **individuals**
2. By specializing classes in the taxonomy of individuals
3. By **instantiating** classes in the taxonomy of **types**
2. By **specializing** classes in the taxonomy of **individuals**

AND

3. By **instantiating** classes in the taxonomy of **types**
4. By specializing classes in the taxonomy of types
There are 2 ways in which you can reuse gUFO properties:

1. By **reusing** gufo properties to make instance-level assertions
2. By **reusing** gufo properties to create type-level cardinality constraints
3. By **specializing** gufo properties
1. By **reusing** gufo properties to make instance-level assertions

- :IbrahimovićBrain gufo:isComponentOf :Ibrahimović
- :Sweden gufo:isMemberOf :EuropeanUnion
- :IbrahimovićFootballSkill gufo:inheresIn :Ibrahimović

*Note that properties that imply existential dependency and part-whole relations are easier to reuse, such as gufo:inheresIn and gufo:isComponentOf*
2. By **reusing** gufo properties to create type-level cardinality constraints
3. By **specializing** gufo properties

```
3. By specializing gufo properties
```

```
REUSING GUFO PROPERTIES (2)
```
1. By specializing gufo properties:
   - Person:Brain:isBrainOf gufo:Object
   - gufo:FunctionalComplex:isComponentOf gufo:Object
   - rdfs:subPropertyOf
   - rdfs:subClassOf
   - rdfs:domain
   - rdfs:range
   - rdf:type
THE TAXONOMY OF INDIVIDUALS AND OBJECT PROPERTIES
TYPES AND INDIVIDUALS

- **Type**: an entity that may be instantiated by (or predicated over) other entities.
  - Also known as “class”, “universal”, “concept”, “kind”, and “category”
  - Person, Movie, Country

- **Individual**: An entity that (unlike a gufo:Type) cannot be instantiated.
  - Also known as “instance”, “particular”, and “object”
  - J.R.R. Tolkien, The Matrix, Brazil

- Every individual must instantiate at least one type in a given point in time.
TYPES

INDIVIDUALS

Person → Baby

Tiago → Camila → Davi

City

Trento → Rome

University

UNIBZ → UNITN

UNIVERSITY

OF

TWENTE.
2ND-ORDER TYPES

1ST-ORDER TYPES

INDIVIDUALS

Tiago
Camila
Davi

Person

gufo:Kind

gufo:Phase

gufo:Object

Baby
2ND-ORDER TYPES
Defined in the taxonomy of types

1ST-ORDER TYPES
Defined in the taxonomy of individuals

INDIVIDUALS
Defined in your dataset
CONCRETE VS ABSTRACT INDIVIDUALS

• Concrete individual
  
  • A gufo:Individual that exists in space-time.
  
  • Concrete individuals comprise:
    • Object-like entities: a car, a mountain, a person, a marriage, a belief
    • Events: a business meeting, a soccer match
    • Situations: the situation in which a person weighs 80 kilograms, the situation in which a bank account is overdrawn

• Abstract individual
  
  • A gufo:Individual that does not exist in space-time in the same way as a gufo:ConcreteIndividual does.
  
  • A gufo:AbstractIndividual has no spatiotemporal qualities in its own right. Hence, it does not make sense to ask how much space it now occupies (Gideon, 2018) and when it was created or destroyed.
  
  • Examples include the number ten, the null set, and the proposition that 'Obama was the president of the United States'.

An entity that (unlike a gufo:Type) cannot be instantiated.

Individuals may be either concrete (e.g., the Earth, Mick Jagger, Brazil, the 1985 Mexico City Earthquake) or abstract (e.g., the number two, the proposition that 'three is a prime number').

Also known as "particular" in the philosophical literature.
A gufo:Individual that exists in space-time.

Concrete individuals comprise not only object-like entities (a car, a mountain, a person, a marriage, a belief), but also events (a business meeting, a soccer match) and situations (the situation in which a person weighs 80 kilograms, the situation in which a bank account is overdrawn).
hasBeginPoint

Identifies the begin point for a gufo:ConcreteIndividual, in the case in which time instants are reified.

In the case of endurants, this identifies the point in time when the endurant comes into existence. In the case of events, this identifies the point in time when the event starts to take place. In the case of situations, this identifies the point in time when the situation begins to hold.

If time instants are not reified, use gufo:hasBeginPointInXSDDate or gufo:hasBeginPointInXSDDateTimeStamp.

Characteristics: hasBeginPoint
- Equivalent To
- SubProperty Of
- Inverse Of
- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive
- Ranges (intersection)
  - ConcreteIndividual
  - Instant

Domains (intersection)
- ConcreteIndividual

Ranges (intersection)
- Instant

Disjoint With
- Equivalent To
- SubProperty Of
- Inverse Of
In the case of endurants, gufo:hasBeginPointInXSDDate determines the time point when the endurant comes into existence. In the case of events, this data property determines the time point when the event starts to take place. In the case of situations, it determines the time point when the situation begins to hold.

Use gufo:hasBeginPoint instead when temporal entities are reified.
gufo:Concrete

rdfs:subClassOf

gufo:Endurant

rdfs:subClassOf

gufo:Object

rdfs:subClassOf

:Car

guco:hasBeginPointInXSDDate

rdfs:domain

xsd:date

rdfs:range
Car rdfs:subClassOf gufo:Concrete Individual

gufo:Endurant rdfs:subClassOf gufo:Object

:Car rdf:type gufo:Endurant

guo:hasBeginPointInXSDDate rdfs:subClassOf rdfs:subClassOf rdfs:domain gufo:hasBeginPointInXSDDate rdfs:range xsd:date

:Car gufo:hasBeginPointInXSDDate "2020-07-01"^^xsd:date
:Car rdf:type gufo:Concrete Individual
guo:hasBeginPointInXSDDate rdfs:domain gufo:Endurant
rdfs:subPropertyOf rdfs:domain gufo:Object
rdfs:subClassOf rdfs:domain :Car
:hasManufacturingDate rdfs:range xsd:date
"2020-07-01"^^xsd:date

:Fiats5001 rdf:type gufo:Concrete Individual
guo:hasBeginPointInXSDDate rdfs:domain gufo:Endurant
rdfs:subPropertyOf rdfs:domain gufo:Object
rdfs:subClassOf rdfs:domain :Car
:hasManufacturingDate rdfs:range xsd:date
"2020-07-01"^^xsd:date
TYPES OF CONCRETE INDIVIDUALS

• **Endurant**
  - A ConcreteIndividual that endures in time and may change qualitatively while keeping its identity.
  - Examples:
    - Ordinary objects of everyday experience, such as a person, a house, and a car;
    - Reified relationships, such as a marriage, a rental contract, and a person's love for another;
    - Existentially-dependent aspects of objects, such as a car's weight, a person's language skills, and a house's color.

• **Event**
  - A ConcreteIndividual that 'occurs' or 'happens' in time. They may be instantaneous or long-running. Events are those "things that happen to or are performed by" (Casati and Varzi, 2015) endurants.
  - Examples:
    - Actions and processes, such as a business meeting, a communicative act, a soccer match, a goal kick
    - Natural occurrences, such as an earthquake, the fall of the meteor that caused the extinction of the dinosaurs.
A gufo:ConcreteIndividual that endures in time and may change qualitatively while keeping its identity.

Examples include: ordinary objects of everyday experience, such as a person, a house, and a car; reified relationships, such as a marriage, a rental contract, and a person's love for another; and existentially-dependent aspects of objects, such as a car's weight, a person's language skills, and a house's color.

Also termed "continuant" in the philosophical literature.
A gufo:Endurant that does not depend on another endurant for its existence (excluding its essential parts and aspects).

Examples of objects include ordinary physical entities, such as a dog, a house, a tomato, a car, Alan Turing, but also socially-defined entities such as The Rolling Stones, the European Union, the Brazilian 1988 Constitution.

Guizziardi (2005) also included the more abstract notion of "Substantial", which generalizes both objects and amounts of matter. That notion was left out from this implementation, together with the notion of amount of matter. Support for the representation of maximally-self-connected amounts of matter is given by gufo:Quantity.
inheresIn

Identifies the gufo:ConcreteIndividual in which the gufo:Aspect inheres. Inherence is a sort of existential dependence. The identified concrete individual is the "bearer" of the aspect.

For example, the color of an object inheres in the object and the average speed of a flight inheres in the flight.
A gufo:IntrinsicAspect that is not measurable.

For example, Bob's belief that the Eiffel Tower is in Paris, his math skills, his headache.

Corresponds to "Mode" in Guizzardi (2005).
ExtrinsicAspect is a gufo:Aspect that depends on one or more concrete individuals. Extrinsic (or 'relational') aspects are reified relationships, e.g., John and Mary's marriage, Mary's employment contract at Nasa, or parts of those relationships, e.g., John's obligations towards Mary in the scope of the marriage, Mary's reciprocal claims, Mary's obligations towards John, John's reciprocal claims. Extrinsic aspects can also be reified one-sided relationships, e.g., John's admiration for Obama (which depends on Obama but does not characterize him).

A gufo:ExtrinsicAspect that connects (involves, mediates) two or more concrete individuals. Relators are reified relationships composed of reciprocal extrinsic modes.

Examples of relators include John and Mary's marriage (composed of John's obligations towards Mary in the scope of the marriage, Mary's reciprocal claims, Mary's obligations towards John, John's reciprocal claims), Mary's employment contract at Nasa, a covalent bond between two atoms.

Equivalent To

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With

Disjoint Union Of
Identifies the endurants mediated by a gufo:Relator.

For example, John and Mary's marriage mediates John and Mary.
A gufo:ExtrinsicAspect that inheres in a concrete individual and depends on others for its existence.

A gufo:ExtrinsicMode can be understood as a reified one-sided relationship, such as John's admiration for Mary.

A gufo:ConcreteIndividual that "occurs" or "happens" in time. They may be instantaneous or long-running. Events are those "things that happen to or are performed by" (Casati and Varzi, 2015) endurants.

Examples include actions and processes, such as a business meeting, a communicative act, a soccer match, a goal kick, the clicking of a mouse button; as well as natural occurrences such as an earthquake, the fall of the meteor that caused the extinction of the dinosaurs.

Also termed "happening", "occurrence", "perduran" or "occurrent" in the philosophical literature.

EVENTS

• Relations between events and endurants:
  • An endurant wasCreatedIn an event
  • An endurant wasTerminatedIn an event
  • An object participatedIn an event
  • An aspect was manifestedIn an event

• Relations between events and situations
  • A situation contributedToTrigger an event
  • An event broughtAbout a situation
Identifies a gufo:Event in which the gufo:Object participated.

Examples include the participation of Freddy Mercury in Queen’s Live Aid Concert and the participation of an airplane in a flight.
Identifies the gufo:Event which brought the gufo:Endurant into existence.

For example, a musical piece is created in an act of composition (or in an event that is part of it), a piece of legislation is created in a complex legislative process. Benevides et al. (2019) only discussed creation of objects; gufo:wasCreatedIn is extended to endurants in general. Further, in that work "createdBy" required the event to "bring about" a situation in which the created object is present. We relax this requirement here, such that the object may be created and terminated in the scope of the identified gufo:Event.
isProperPartOf

http://purl.org/nemo/gufo#isProperPartOf

Annotations isProperPartOf

Annotations

rdfs:label [language: en] isProperPartOf

rdfs:comment [language: en]
Identifies a whole of which the entity is a proper part.

gufo:isProperPartOf is the most generic part-whole relation in this implementation. Use the various sub-properties provided in order to represent specific types of part-whole.

Characteristics: isProperPartOf

- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive

Description: isProperPartOf

- Equivalent To
- SubProperty Of
- Inverse Of
- Domains (intersection)
- owl:Thing
- Ranges (intersection)
- owl:Thing
- Disjoint With
- SuperProperty Of (Chain)
QUALITIES AND DATATYPES
QUALITIES

- We distinguish between the color of an apple from the particular shade of red it has at some point in time.

- This allows us to:
  - Express that the color of the apple changes
  - Represent the value of the color in multiple measurement systems
  - Represent the truth maker of comparative relations
Apple X

Color of Apple X inheres in

(0, 92, 100)

(255, 20, 20)
Apple X

inheres in

Color of Apple X

(224, 100, 90)

(0, 60, 230)
There are 3 ways to represent qualities in gUFO:

1. By **specializing** the datatype property `gufo:hasQualityValue`
2. By **specializing** the object property `gufo:hasReifiedQualityValue`
3. By **specializing** the class `gufo:Quality`
   a. that is projected in a **1-dimensional** space
   b. that is projected in a **n-dimensional** space

Choosing between these options depends mostly on your use case requirements!
1. By **specializing** the datatype property **gufo:hasQualityValue**
2. By specializing the object property gufo:hasReifiedQualityValue

REPRESENTING QUALITIES (2)
3a. By **specializing** the class **gufo:Quality** that is projected in a **1-dimensional** space
We may want to impose cardinality constraints on colors:
3b. By **specializing** the class **gufo:Quality** that is projected in a **n-dimensional** space.
We can also declare the dimensions in which a quality type can be projected into.
THE TAXONOMY OF TYPES

05
In 1970

- Brazilian
- Adult
- Man
- Football player
- Philanthropist
- Person
- Husband
- Actor
- Father

**Check**: Minister of sports - **X**
In 1994

- Brazilian
- Adult
- Minister of sports
- Actor
- Husband
- Person
- Man
- Father

Red X: Not applicable
Green Check: Applicable
In 2020

- Brazilian
- Adult
- Man
- Father
- Husband
- Philanthropist
- Person

- Football player
- Actor
- Minister of sports

(All roles not confirmed in 2020)
RIGIDITY

- A metaproperty regarding the instantiation dynamics between types and their instances
  - Rigid types: Person, Man
  - Anti-rigid types: Adult, Father, Husband, Football Player
  - Semi-rigid types: Brazilian

- Originally proposed in the OntoClean methodology

RIGID TYPES

- Essentially classify its instances

Pelé is both a **Person** and a **Man** in every possible point in time in which he exists (even counterfactual ones)
ANTI-RIGID TYPES

• Contingently classify its instances

Pelé was contingently a Child and an Adult. Now he is a Senior.
SEMI-RIGID TYPES

• Essentially classify some of its instances and contingently classify others

Pelé is a natural born Brazilian, so it is essential for him to be so.

Meligeni became a Brazilian when he was a child. Thus, being so is an accidental property for him.
RIGIDITY IN GUFO

- Type
  - Abstract Individual Type
  - Concrete Individual Type
    - Event Type
    - Situation Type
    - Endurant Type
  - Rigid Type
    - Kind
    - Category
    - Subkind
  - Non-Rigid Type
    - Anti-Rigid Type
      - Role
      - RoleMixin
    - Phase
    - PhaseMixin
  - Semi-Rigid Type
    - Mixin

- rdf:type
  - Person
  - Living Being
  - Man
  - Footballer
  - Customer
  - Child
  - Infant
  - Music Artist
WHY ARE THESE DISTINCTIONS USEFUL?

• They allows us to:
  • Properly characterize the various types in our domain
  • Create consistent taxonomies

• gUFO leverages these distinctions to define rules to help us design better models!
A **rigid** type can be specialized by a **rigid** type.

A **rigid** type can be specialized by an **anti-rigid** type.
An **anti-rigid** type can be specialized by an **anti-rigid** type.

A **semi-rigid** type can be specialized by an **semi-rigid** type.
A **semi-rigid** type can be specialized by an **anti-rigid** or a **rigid** type.
An **anti-rigid** type cannot be specialized by a **rigid** type

An **anti-rigid** type cannot be specialized by a **semi-rigid** type
Are these the same statue?
What about now?
IDENTITY CRITERIA

• A “function” that allows us to distinguish and count individuals

• It helps us to answer questions like:
  • “Is that my dog?”
  • “Is this the same actor I have seen in that other movie?”

• It defines how much an individual can change and remain the same

• Every individual adheres to an identity criteria!

IDENTITY CRITERIA

- Consider the following scenario:
  - **time duration**: 1 hour, 2 hours...
  - **time interval**: “1:00 – 2:00 next Tuesday”, “2:00 – 3:00 next Wednesday”

- Would making time interval a subclass of time duration be a good modeling decision?
  - 2 durations are the same if they have the same length
  - 2 intervals are the same if they occur at the same time
SORTALITY

• A metaproperty regarding the relation between types and identity criteria:

• **Sortal type:** all of its instances follow the same identity criteria
  • Person, Man, Student, Adult, Marriage

• **Non-sortal type:** its instances follow different identity criteria
  • Agent, Customer, Physical Object

• **Ultimate sorta type:** provides the identity criteria to its instances
  • Person, Organization, Marriage
SORTALITY IN GUFO

- **Type**
  - Abstract Individual Type
  - Concrete Individual Type
    - Event Type
    - Situation Type
    - Endurant Type
      - Sortal
        - Kind
        - Subkind
        - Role
        - Phase
      - NonSortal
        - Category
          - RoleMixin
          - PhaseMixin
          - Mixin

```
[Person rdf:type = Customer]
[Man rdf:type = Footballer]
[Footballer rdf:type = Footballer]
[Child rdf:type = Footballer]
[Living Being rdf:type = Customer]
[Customer rdf:type = Customer]
[Infant rdf:type = Customer]
[Music Artist rdf:type = Customer]
```
A sortal type cannot specialize multiple ultimate sortal types

An ultimate sortal type cannot specialize another ultimate sortal type
A sortal type must specialize an ultimate sortal type.

An ultimate sortal type cannot specialize another ultimate sortal type.
These classes provide identity criteria to their instances. The instances of these classes follow different identity criteria. The instances of these classes follow the same identity criterion, which is inherited from Person.
• **Rigid sortal types** that supply an identity principle to its instances

• Also known as **natural kinds** in the philosophical literature

• The basic types of things that exist in our domain of interest
• **Rigid sortal types** that (indirectly) specialize ultimate sortal types (e.g. kinds), from who they inherit the identity criteria

• Subkinds specialize kinds or other subkinds
• Subkinds are often defined in partitions
CATEGORIES

- **Rigid non-sortal types** that capture essential properties of individuals that instantiate different kinds
- They usually represent the most abstract layer of an ontology
- They generalize sortal types
- They do not have direct instances
A type $T$ is relationally dependent on a type $P$ by means of a relation $R$

$$\forall x \ T(x) \rightarrow \exists y. \ P(y) \land R(x,y)$$

This type of dependency is known as generic dependency

Examples:

- **Student** depends on **School**
- **Author** depends on **Book**
- **Father** depends on **Offspring**
ROLES

- **Anti-rigid relationally dependent sortal** types
A person plays the role of student when she studies at a school.

Roles must specialize an ultimate sortal (e.g.), from which they inherit their identity criteria.

The relational dependence is represented by the mandatory relation.
How do we model the customer role if it is playable by both people and organizations?
We should use the **RoleMixin Pattern**!
(also known as role with disjoint allowed types)
Alternative version

gufo:Category

rdfs:subClassOf

:Agent

rdf:type

gufo:RoleMixin

rdfs:subClassOf

:Customer

rdf:type

:Person

rdfs:subClassOf

:Organization

rdf:type

gufo:Kind

rdf:type
ROLEMIXINS

- An anti-rigid relationally dependent non-sortal type
- Examples:
  - Customer and buyer are roles playable by people and organizations
  - Trustee is a role playable by people or objects
PHASES

- **Anti-rigid sortal** types whose instantiation are characterized by changes in intrinsic properties of their instances
- Phases always come in partitions (disjoint and complete)
- Examples:
  - Child, Adult, and Elder are phases of a Person
  - Functioning and Broken are phases of a Car
PHASEMIXINS

- **Anti-rigid non-sortal** types whose instantiation are characterized by changes in intrinsic properties of its instances

- Simply put, a **non-sortal phase**
MIXINS

- **Semi-rigid non-sortal** types
- Capture properties that are essential to some individuals and accidental to others
- Can be specialized by anti-rigid and rigid types
HISTORICAL DATA

- By default, there is no support for representing change in the Semantic Web.

- Then, what do we do when:
  - a person loses/gains weight?
  - a rental car is under repair?
  - a band changes members?
  - a student graduates?
  - a president leaves office?
CHANGING RELATIONSHIPS

- Change in relationships represented via its truthmakers are natively supported via their begin and end point properties.

```
<Person rdf:type gufo:mediates gufo:hasBeginPointInXSDDate "2017-01-01"^^xsd:date gufo:hasEndPointInXSDDate "2019-08-05"^^xsd:date .

:Employment rdf:type gufo:mediates .

:Organization rdf:type .

:John rdf:type .

:John's Employment rdf:type .

:UTwente rdf:type .
```

- Endurant
  - Aspect
    - IntrinsicAspect
    - IntrinsicMode
    - ExtrinsicAspect
    - ExtrinsicMode
- Situation
  - QualityValueAttributionSituation
  - TemporaryConstitutionSituation
  - TemporaryInstantiationSituation
  - TemporaryParthoodSituation
  - TemporaryRelationshipSituation
John's past employment

:Person

rdf:type

:John

gufo:mediates

:John's Employment @ UT

:Employment

rdf:type

:John's Employment @ UT

gufo:mediates

:John's Employment @ UNIBZ

:Employment

gufo:mediates

:John's Employment @ UNIBZ

gufo:mediates

:UTwente

rdf:type

:UTwente

gufo:mediates

:John's current employment

"2017-01-01"^^xsd:date

:John's Employment

gufo:mediates

:John's Employment

gufo:mediates

:UTwente

"2019-08-05"^^xsd:date

:John's Employment @ UT

gufo:mediates

:John's Employment @ UNIBZ

"2019-08-06"^^xsd:date

:John's Employment @ UNIBZ
OTHER CHANGES

- Changes regarding:
  - Instantiation (John became a professor)
  - Quality value attribution (John’s salary changed)
  - Part-whole relations (John switched his car tires)
  - Temporary relations (John no longer is friends with Paul)

- Are all captured via specific subclasses of **gufo:Situation**
  - A gufo:Concretelndividual that is a particular configuration of a part of reality which can be understood as a whole and in which entities stand in relations.
  - A situation may be counterfactual or actual. An actual situation (or in other words, a "fact") "obtains" in a certain time instant or during a time interval.
Temporary quality value attribution

:Person
  rdf:type
  :John
  gufo:mediates
  :John's Employment
  gufo:mediates
  :Organization
  rdf:type

:Employment
  rdf:type
  :John's Employment
  gufo:mediates
  :UTwente
  rdf:type

:Organization
  rdf:type

:John
  gufo:mediates
  :John's Employment

:Job's Salary in 2017
  gufo:standsInQualityValueAttributionSituation
  gufo:hasBeginPointInXSDDate
  "2017-01-01"^^xsd:date
  gufo:hasEndPointInXSDDate
  "2017-12-31"^^xsd:date
  gufo:ConcernsSalaryValueInEur
  "1000"^^xsd:double
  (sub gufo:ConcernsQualityValue)

Unversity of Twente
Temporary instantiation situation

:Person rdf:type :Footballer rdfs:subClassOf "2000-03-15"^^xsd:date gufo:hasBeginPointInXSDDate gufo:standsInQualityInstantiationSituation :John rdf:type gufo:TemporaryInstantiationSituation :JohnIsA :Footballer gufo:hasEndPointInXSDDate "2021-10-22"^^xsd:date gufo:concernsNonRigidType gufo:standsInQualityInstantiationSituation

UNIVERSITY OF TWENTE.
CONCLUSION
CONCLUSIONS

• We need all the help we can get!
  • Rules
  • Reuse
  • Foundational patterns
  • Automation of quality control

• We brought the benefits that were only available to OntoUML users to Semantic Web implementers

• Better integration between the taxonomy of types and taxonomy of individuals than in OntoUML (due to limitations of UML)
HOW ABOUT EXPRESSIVENESS

• OWL 2 DL fragment employed

• But less expressive fragments possible
  • Application-dependent choices on what restrictions to leave out
  • E.g., punning can be ignored or replaced by annotation properties

• Rules that cannot be expressed in OWL are implemented in the plugin
  • But can be expressed as shape constraints: SHACL
HOW DOES GUFO FIT IN THE OVERALL UFO/ONTOUML ECOSYSTEM?

- OntoUML to gUFO-based OWL transformation
  - incorporated in OntoUML Visual Paradigm plugin
- Using OntoUML as a starting point gives access to simulation, antipattern detection
- gUFO-based Ontology-Based Data Access (OBDA)
  - high-level access to relational data
ONGOING AND FUTURE WORK

• We want to port the engineering tools we developed for OntoUML into gUFO
  • Pattern-based development in the Protégé plugin
  • Anti-pattern detection
  • Simulation

• Reverse engineering OWL ontologies to OntoUML

• gUFO-based implementations of UFO-based reference ontologies:
  • gUFO-C: Intentional and Social Layer
  • gUFO-L: Core Ontology of Legal Aspects
  • gUFO-S: Core Ontology of Services
IMPLEMENTING BETTER ONTOLOGIES WITH GUFO

TIAGO PRINCE SALES
T.PRINCESALES@UTWENTE.NL