RDF, RDFS & JSON-LD

UROŠ KRČADINAC

EMAIL: uros@krcadinac.com

URL: uros@krcadinac.com

Linked Data

- Linked Data is a way to create a network of standards-based machine interpretable data across different documents and Web sites.
- It allows an application to start at one piece of Linked Data, and follow embedded links to other pieces of Linked Data that are hosted on different sites across the Web.

Reference taken from: http://www.w3.org/TR/json-ld/

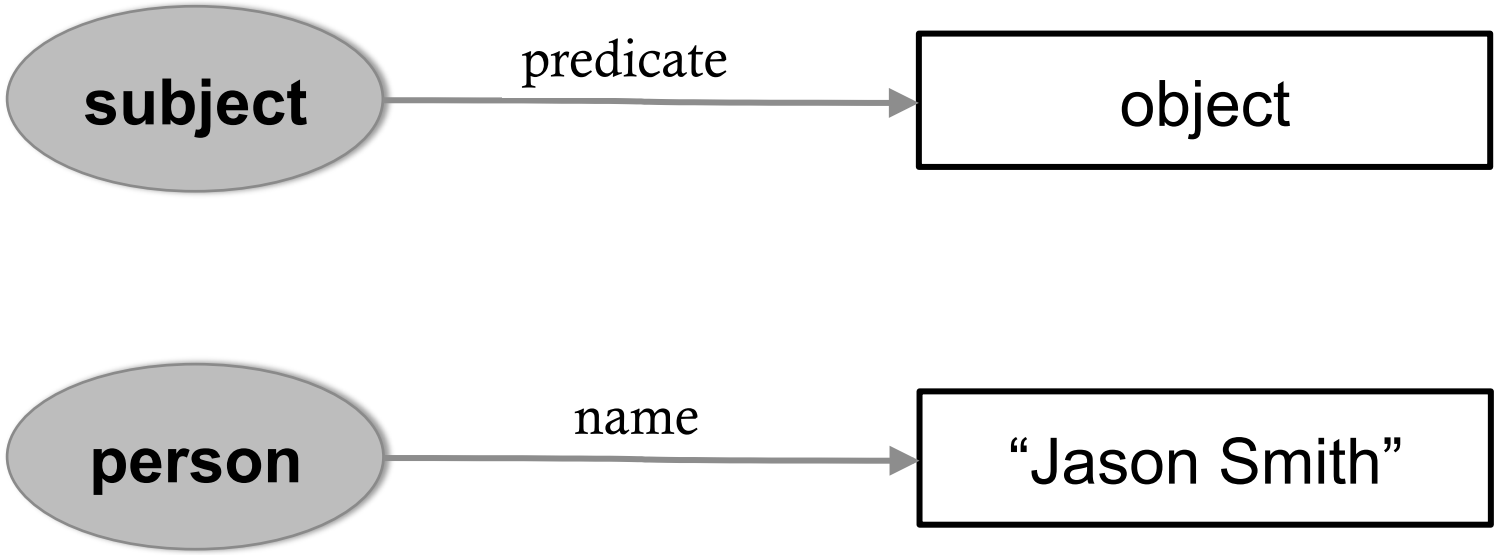
What is RDF?

- Resource Description Framework
- W3C standard for describing data on the Web
- One of the three fundamental Semantic Web technologies (other two being SPARQL and OWL)

What is RDF?

- RDF is the data model of the Semantic Web
- Simple model, based on a graph
- Describes relations between things

Example of RDF graph



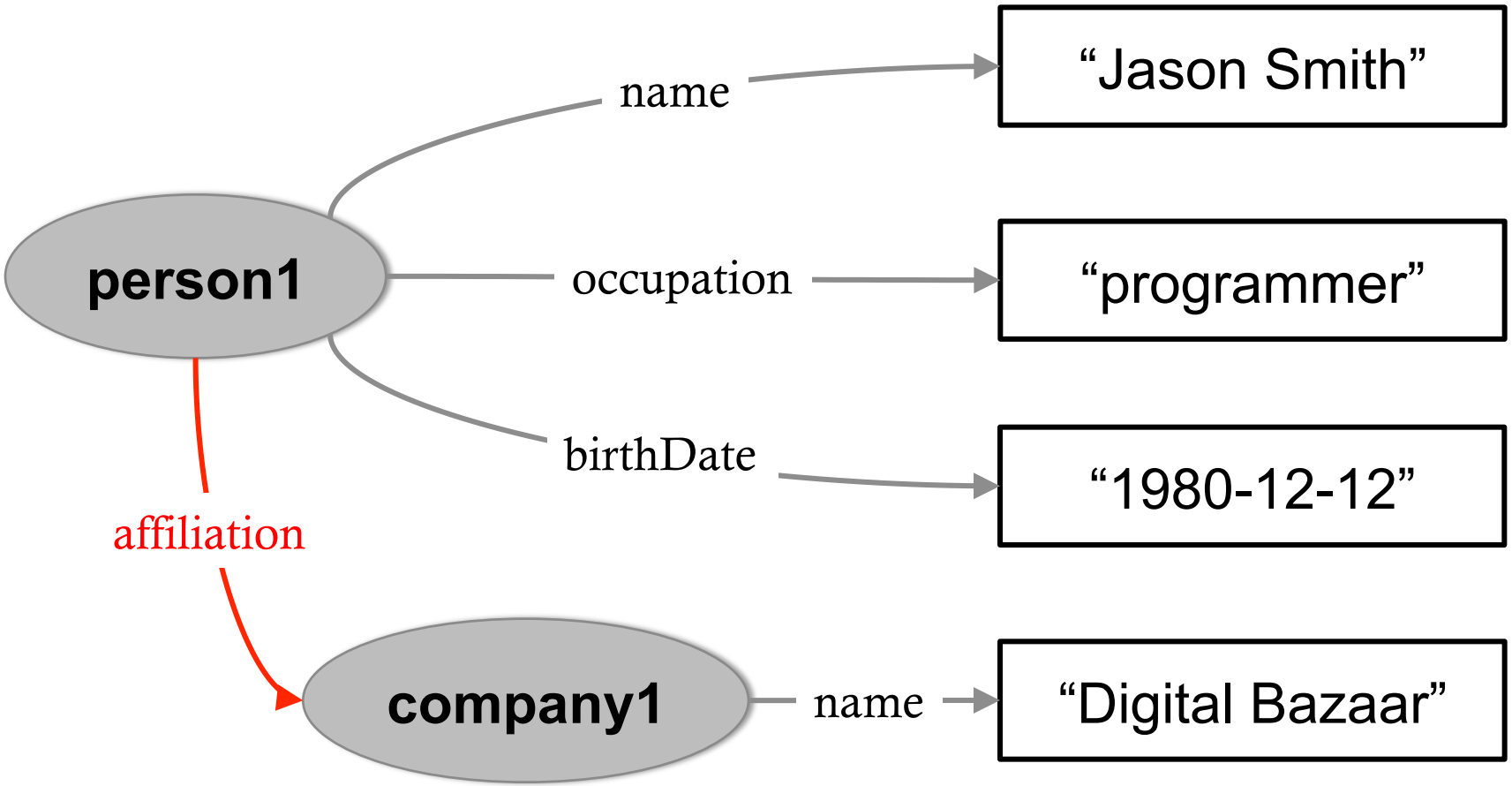
RDF is a graph

- RDF is based on triplets

  (subject predicate object)

- Graph elements
  - Node (for describing subjects and objects)
    - Resource (depicted with an elipse)
    - Literal (depicted with a rectangle)
  - Relationship (for describing predicates)

Multiple properties

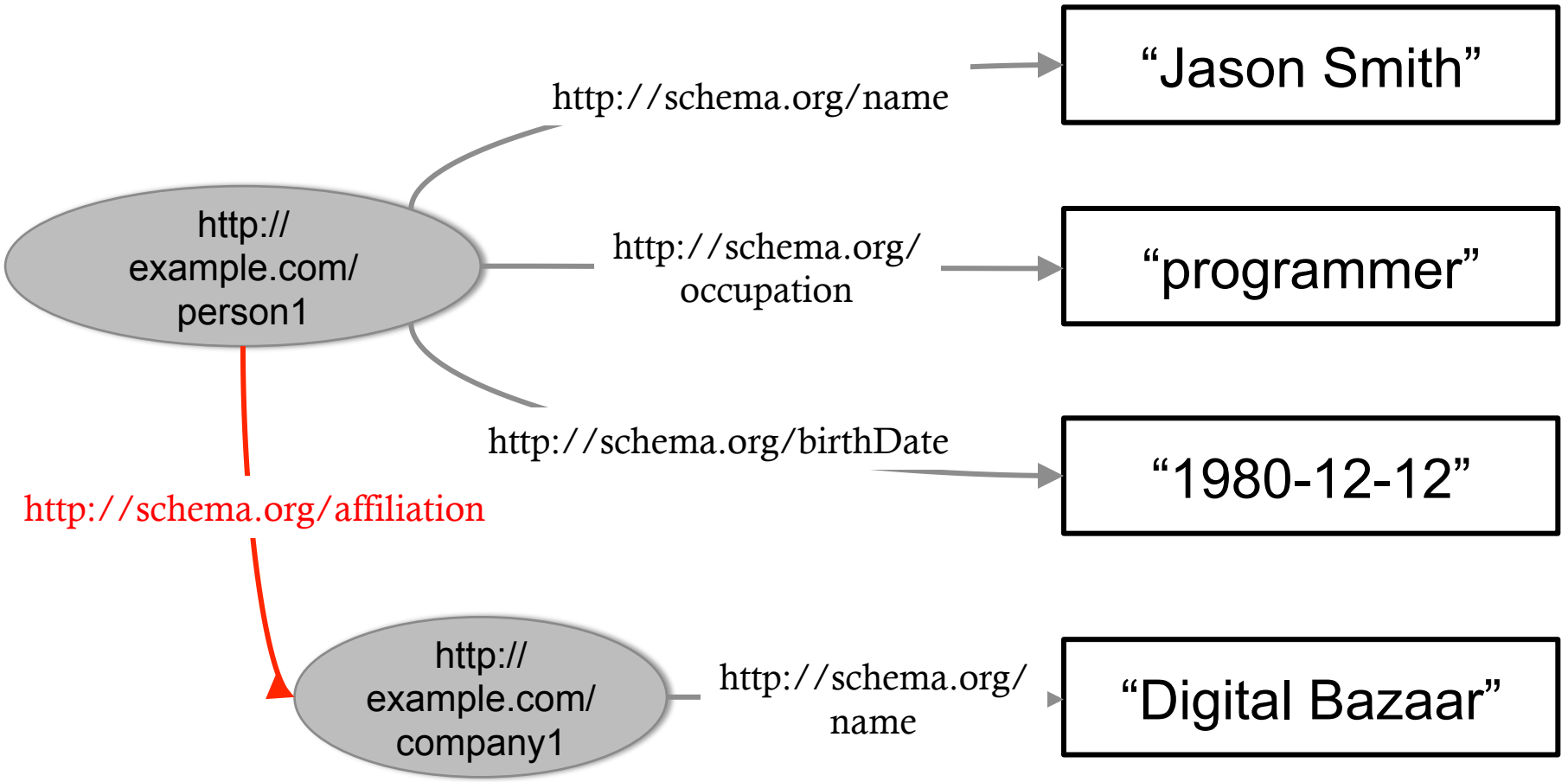


Using Web infrastructure

- In a *global data repository* on the Web we must identify resources globally and uniquely
- URI
- Data naming using URIs, usually with protocol http:// - **THIS IS A KEY CONCEPT FOR LINKED DATA**

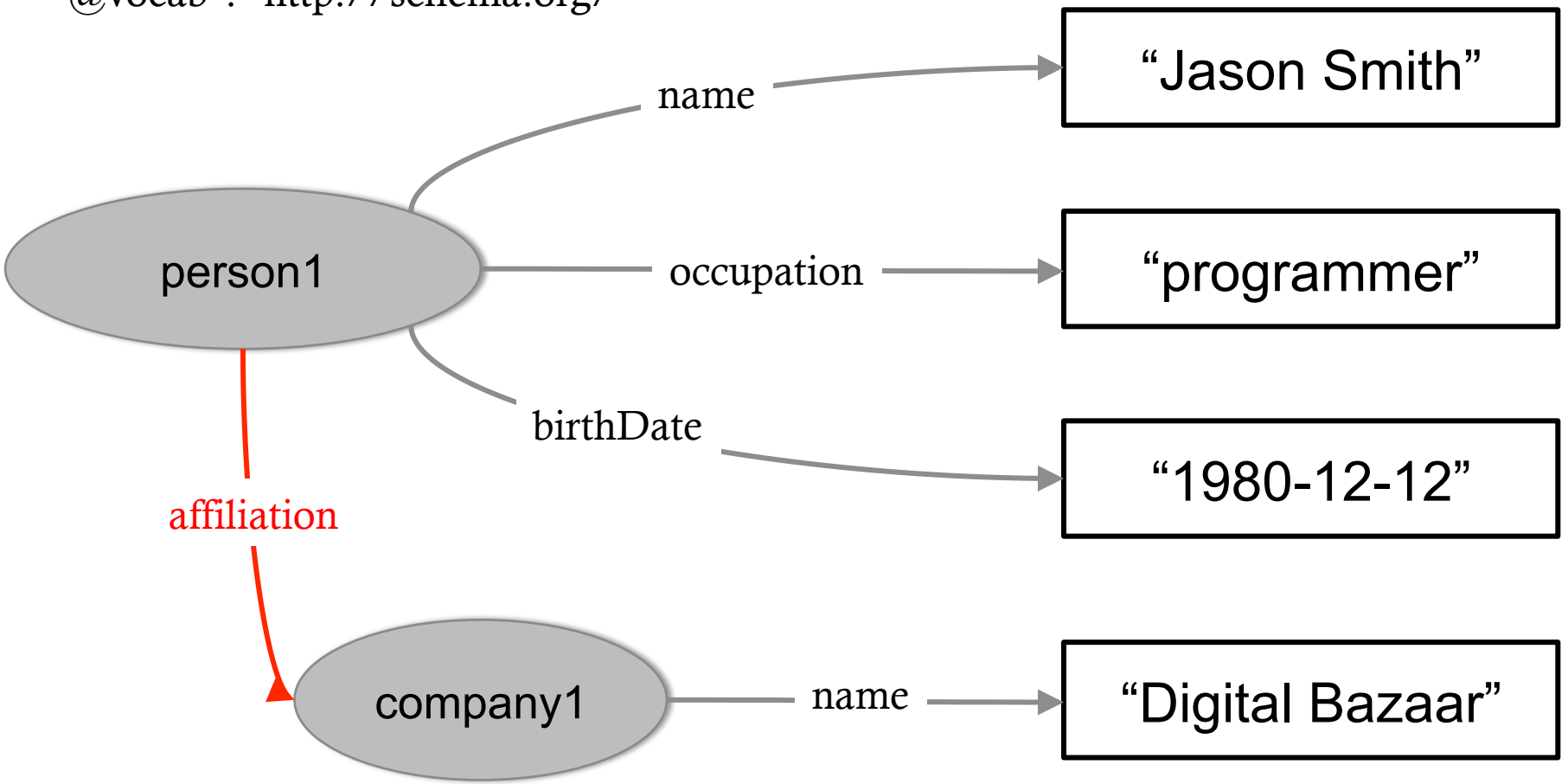
URL – Uniform Resource Locator	location
URI – Uniform Resource Identifier	identifier
IRI – International Resource Identifier	identifier

Graphs can have named resources



Using vocabularies

"@vocab": "http://schema.org/"



Triplet form

person1	name	“Jason Smith” .
person1	occupation	“programmer” .
person1	birthDate	“1980-12-12” .
company1	name	“Digital Bazaar” .
person1	affiliation	company1 .

Simple Rules

- Use URIs to identify resources
- If the same URI is used on multiple places, then they identify the same resource
- This enables for easy interlinking of data coming from multiple sources

RDFS

UROŠ KRČADINAC

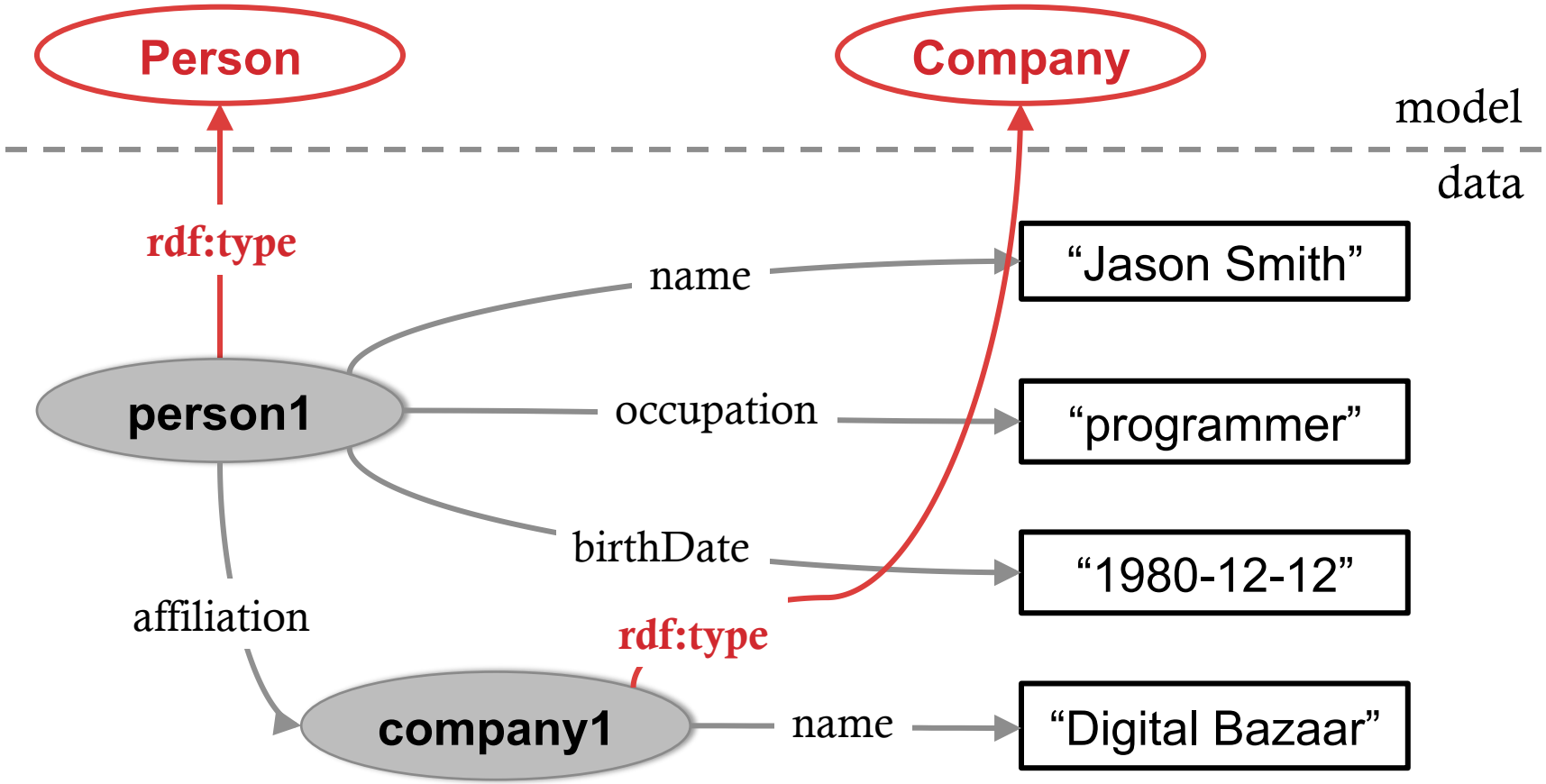
EMAIL: uros@krcadinac.com

URL: uros@krcadinac.com

RDFS

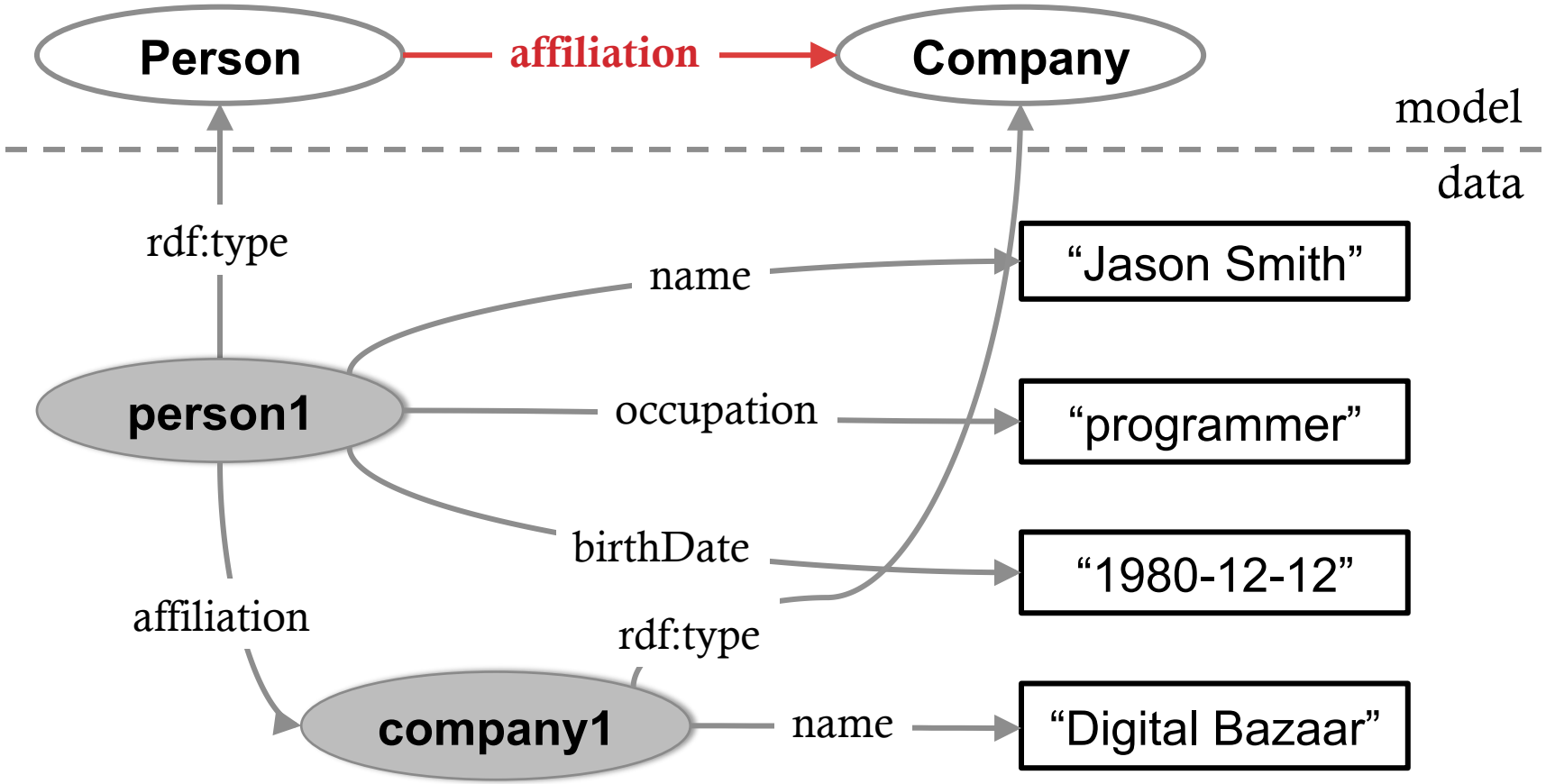
- RDFS - RDF Schema
- Adding semantics to RDF
- Creating data schema – vocabulary
- Vocabulary is defined in the same way as data

Classes and Hierarchies

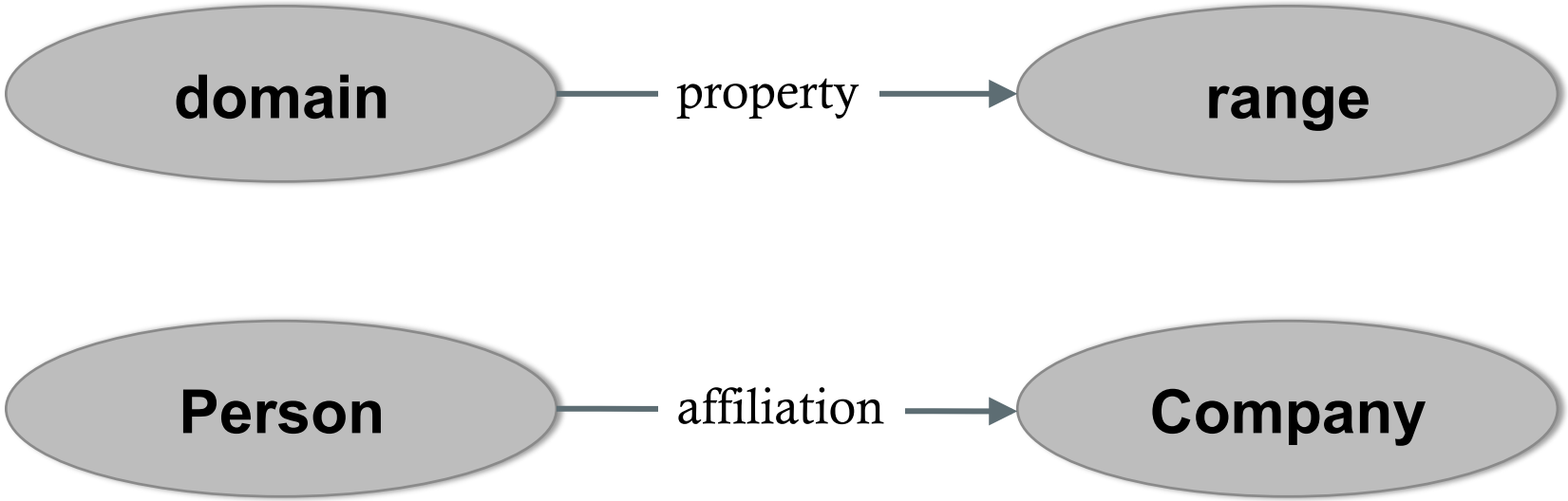


Person rdf:type rdfs:Class .
person1 rdf:type Person .

Properties


Person
affiliation
Company
model
data
rdf:type
name
"Jason Smith"
person1
occupation
"programmer"
birthDate
"1980-12-12"
affiliation
rdf:type
company1
name
"Digital Bazaar"

Properties



affiliation rdf:type rdf:Property .
affiliation rdfs:domain Person .
affiliation rdfs:range Company .

Properties

Domain points to a class (or multiple classes) a property can be used on

Range represents a class (or multiple classes) that can be a value of a property

Both domain and range are optional. If domain is not defined, property can be used on any class. If range is not defined, value of a property can be any class.

Not the same as with OO languages

- Properties can exist without any class, they are first class citizens
- Properties can be extended, they can have their own hierarchy of sub-properties
- Properties can not be overridden on a lower level of hierarchy (by sub-properties)

Schema.org

Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond. Schema.org is sponsored by Google, Microsoft, Yahoo and Yandex.

The vocabulary cover entities, relationships between entities and actions, and can easily be extended through a well-documented extension model

Schema.org

Some of the class defined:

- Creative works: CreativeWork, Book, Movie, MusicRecording, Recipe, TVSeries …
- Netekstualni objekti: AudioObject, ImageObject, VideoObject
- Event
- Organization
- Person
- Place, LocalBusiness, Restaurant …
- Product, Offer, AggregateOffer
- Review, AggregateRating
- Action

Schema.org

Some of the extensions:

- auto.schema.org
- bib.schema.org

RDF(S) vocabulary

RDF and RDFS vocabularies

Prefixes: **rdf** i **rdfs**

Classes (some of them)

- rdfs:Class
- rdfs:Property
- rdfs:Literal

Properties (some of them)

- rdf:type (resurs je instanca neke klase)
- rdfs:subClassOf (class is a subclass of another class)
- rdfs:subPropertyOf (subproperty)
- rdfs:seeAlso (reference to a description)
- rdfs:domain (domain of a property)
- rdfs:range (range of a property)

JSON – JavaScript Object Notation

- Lightweight format for data exchange
- Simple
  - For the developers writing it
  - For the machines processing it
- JSON is a text-based format
- Independent from the programming language

JSON object

- A set of name-value pairs
- JSON object starts with an open brace ({), and ends with a closing brace (})
- Name and value are separated by colon (:), and name/value pairs are separated with comma (,)

JSON object example

```
{
    “title” : “The Matrix”,
    “producer” : “Joel Silver”,
    “release_year” : 1999
}
```

JSON array

- JSON array represents an ordered sequence of values.
- Starts with an opening bracket [, and ends with a closing bracket]
- Values are separated by comma

JSON array example

```
[
    {
        "title" : "The Matrix",
        "producer" : "Joel Silver",
        "release_year" : 1999
    },
    {
        "title" : "Equilibrium",
        "producers" : [
            {
                "name" : "Joel Silver"
            },
            {
                "name":"Lucas Foster"
            }
        ],
        "release_year" : 1999
    }
]
```

JSON-LD syntax

- Syntax for serializing RDF data into JSON format
- JSON-LD is primarily intended to be a way to use Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines (MongoDB, ElasticSearch, etc.)
- It can be combined with other Semantic Web technologies (e.g. SPARQL)

JSON-LD

In addition to all the features JSON provides, JSON-LD introduces:

- a universal identifier mechanism for JSON objects via the use of IRIs
- a way to disambiguate keys shared among different JSON documents by mapping them to IRIs via a context
- a mechanism in which a value in a JSON object may refer to a JSON object on a different site on the Web
- the ability to annotate strings with their language

Keywords

@id – Used to uniquely identify things that are being described in the document with IRIs or blank node identifiers

@type – Used to set the data type of a node

@context – Used to define the short-hand names that are used throughout a JSON-LD document. These short-hand names are called terms

@language – Used to specify the language for a particular string value

JSON document example

```
{
        "name": ”Jason Smith",
        ”homepage": "http://jason.smith.org/",
        "image": "http://jason.smith.org/images/jason.png”
}
```

JSON-LD document exam

```
{
        "http://schema.org/name": ”Jason Smith",
        "http://schema.org/url": {
                "@id": "http://jason.smith.org/”
        },
        "http://schema.org/image": {
                "@id": "http://jason.smith.org/images/jason.png”
        }
}
```

The '@id' keyword means 'This value is an identifier that is an IRI'

Every property is unambiguously identified by an IRI (like `name`, `url` and `image`). Developers, and programs, can follow the IRI address and look up the property definition.

This process is called **IRI dereferencing**.

Using @context element

@context is used to map terms to IRIs

```
{
    "@context": {
        "name": "http://schema.org/name",
        "image": {
            "@id": "http://schema.org/image",
            "@type": "@id"
        },
        "homepage": {
            "@id": "http://schema.org/url",
            "@type": "@id"
        }
    }
}
```

This means that 'name' is shorthand for 'http://schema.org/name'

This means that 'image' is shorthand for 'http://schema.org/image'

This means that a string value associated with 'image' should be interpreted as an identifier that is an IRI

This means that 'homepage' is shorthand for 'http://schema.org/url'

This means that a string value associated with 'homepage' should be interpreted as an identifier that is an IRI

Defining @context inline

```
{
     "@context": {
          "name": "http://schema.org/name",
          "image": {
               "@id": "http://schema.org/image",
               "@type": "@id"
          },
          "homepage": {
               "@id": "http://schema.org/url",
               "@type": "@id"
          }
     },
     "name": ”Jason Smith",
     "homepage": "http://jason.smith.org/",
     "image": "http://jason.smith.org/images/jason.png"
}
```

Defining @context externally

```
{
        "@context": "http://json-ld.org/contexts/person.jsonld",
        "name": ”Jason Smith",
        "homepage": "http://jason.smith.org/",
        "image": "http://jason.smith.org/images/jason.png"
}
```

Defining the context in an document allows for reuse of the document definition and term to IRI mappings.

Referencing @context via HTTP Link attribute

JSON-LD context (@context) can be defined in the HTTP header, by using the **Link** attribute.

```
GET /jason-smith.json HTTP/1.1
Host: example.com
Accept: application/ld+json,application/json,*/*;q=0.1
====================================
HTTP/1.1 200 OK
…
Content-Type: application/json
Link: <http://json-ld.org/contexts/person.jsonld>; rel="http://www.w3.org/
ns/json-ld#context"; type="application/ld+json"
{
        "name": ”Jason Smith",
        "homepage": "http://jason.smith.org/",
        "image": "http://jason.smith.org/images/jason.png"
}
```

Defining resource type (class)

The type of a particular node can be specified using the **@type** keyword. Types are uniquely identified with an IRI.

```
{
        ...
        "@id": "http://example.org/places#BrewEats",
        "@type": "http://schema.org/Restaurant",
        ...
}
```

Defining resource type (class)

A node can be assigned more than one type by using an array:

```
{
        ...
        "@id": "http://example.org/places#BrewEats",
        "@type": [
                "http://schema.org/Restaurant",
                "http://schema.org/Brewery”
        ],
        ...
}
```

Defining resource type (class)

The value of an @type key may also be a term defined in the active context:

```
{
     "@context": {
          ...
          "Restaurant": "http://schema.org/Restaurant",
          "Brewery": "http://schema.org/Brewery"
     },
     "@id": "http://example.org/places#BrewEats",
     "@type": [
          "Restaurant",
          "Brewery”
     ],
     ...
}
```

Defining vocabulary

If all properties and types may come from the same vocabulary, keyword **@vocab** allows for defining the common prefix for all terms.

```
{
        "@context": {
                "@vocab": "http://schema.org/"
        },
        "@id": "http://example.org/places#BrewEats",
        "@type": "Restaurant",
        "name": "Brew Eats"
        ...
}
```

Compact IRI

A compact IRI is a way of expressing an IRI using a prefix and suffix separated by a colon (:).. E.g. if we want to use the FOAF vocabulary (http://xmlns.com/foaf/0.1/), we can introduce the prefix `foaf`.

```
{
    "@context": {
        "foaf": "http://xmlns.com/foaf/0.1/"
        ...
     },
     "@type": "foaf:Person”,
     "foaf:name": "Dave Longley”,
     ...
}
```

`foaf:name` is expanded into IRI http://xmlns.com/foaf/0.1/name

`foaf:Person` is expanded into IRI http://xmlns.com/foaf/0.1/Person

Example 1

There is a class *Person*.

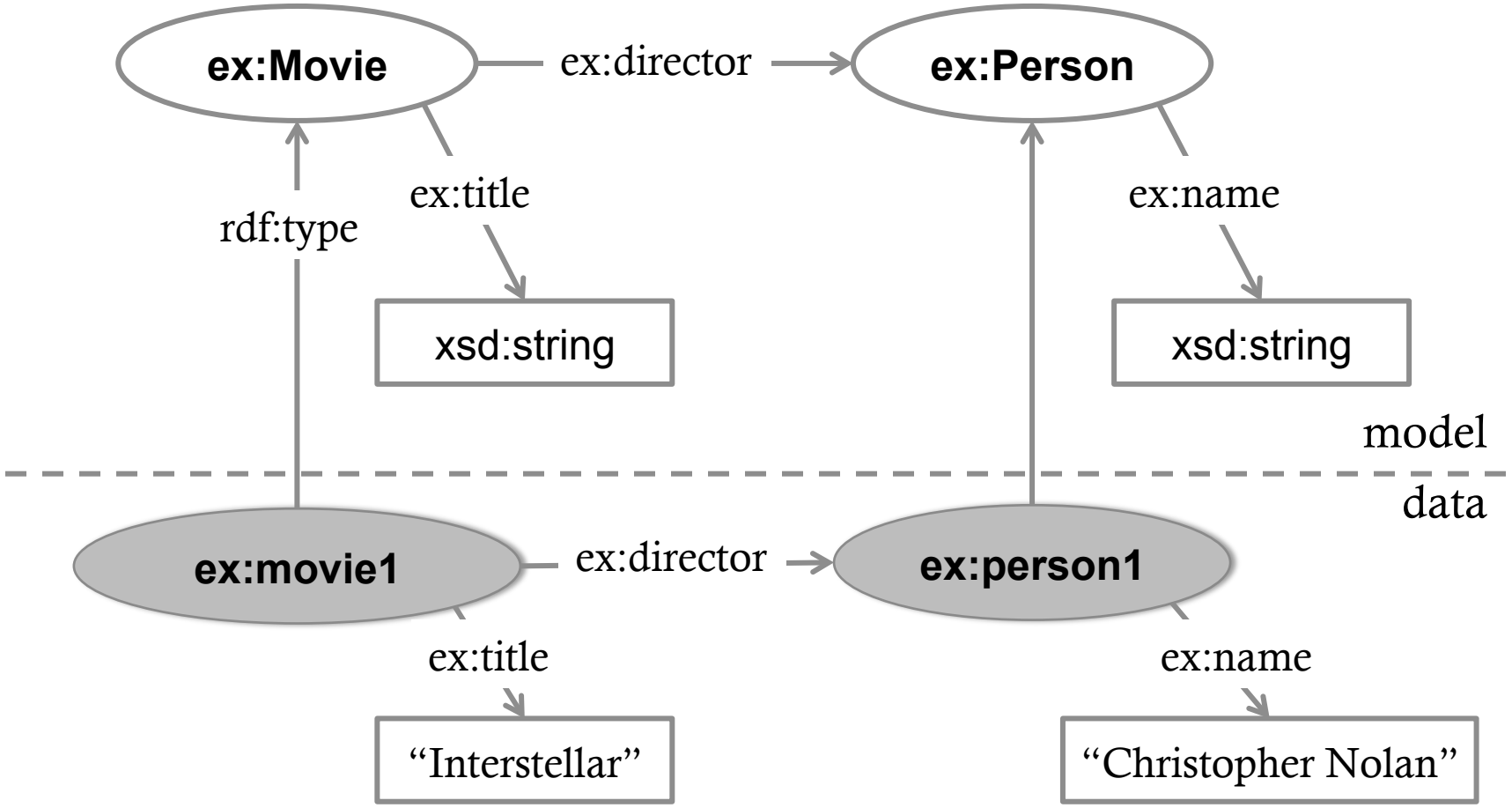
Person can have an attribute *name*.

There is class Movie.

Movie has an attribute *title* that is a string and an attribute *director* which is a person who directed the movie.

There is a movie titled “Interstellar”. The movie director is Christopher Nolan.

Example 1 - Graph



Example 1 – JSON-LD

```
{
        "@context:": {
                "@vocab": "http://example.com/"
        },
        "@id": ”http//example.com/moveie1",
        "@type": ”Movie",
        ”title": "Interstellar",
        ”director": {
                "@type": ”Person",
                "@id": ”http://example.com/person1",
                "name": "Christopher Nolan"
}
```

DC - Dublin Core

DC - Dublin Core (Metadata Initiative)

Idea – describing documents by using set of RDF elements

- Predefined vocabulary
- Enables describing data such as: author, co-author, title, topic, date created…
- Free access to this metadata and their interlining over multiple sources

DC - Dublin Core

Prefix: **dc**

Contains no classes, only properties

Properties (some):

- dc:creator (author)
- dc:contributor (somebody who contributed, but is not an author)
- dc:date
- dc:description
- dc:language
- dc:publisher
- dc:subject
- dc:title

FOAF

FOAF - Friend Of A Friend

Idea:

- Describe basic information about people (name, lastname, email address, homepage...)
- Link people who know each other (knows)
- No data silos like with social networks

FOAF

Prefix: **foaf**

Classes (some of them)

- foaf:Person
- foaf:OnlineAccount

Properties (some of them)

- foaf:name
- foaf:firstName
- foaf:lastName
- foaf:nick (nickname)
- foaf:mbox (mailbox)
- foaf:knows
- foaf:homepage
- foaf:workplaceHomepage
- foaf:account
- foaf:accountName
- foaf:accountServiceHomepage
- foaf:depiction (depiction of a specific resource)

Questions?

UROŠ KRČADINAC

EMAIL: uros@krcadinac.com

URL: uros@krcadinac.com