26-March-2014

Roles in Schema.org [work in progress]

Overview

Historically, schema.org descriptions have been simple and unqualified. For example, the <http://schema.org/actor> property relates a Movie, TVEpisode, RadioSeries etc. to a Person. Although this simplicity was well motivated, and appropriate to schema.org's emphasis on ease of adoption for publishers and webmasters, there are many scenarios when it is useful to provide a richer description. This document presents a data structure and supporting vocabulary for such situations, motivated by practical considerations around sports and other schemas.

Status: this is a proposal - schema.org would like feedback on the design, including choice of terminology. The appendix has notes on similar related designs. *Where possible we use current real schema.org vocabulary; but in a few cases we might introduce convenient but fictional properties for ease of explanation.*

Example scenarios

1. **Sports:** modeling time-bounded sporting roles.

In describing sports-related information, it is common to want to temporarily qualify information.

For example: **JoeMontana** was an **athlete** in the ***FootballTeam*** **SF49ers**, between a **startDate** of **2002** and an **endDate** of **2008**; his **position** was that of **QuarterBack**.

2. **Media**: additional details about the relationship between two entities, e.g. an actor and a movie.

Simple schema.org already allows us to say that **Bill Murray** was an actor in the ***Movie*** **Ghostbusters**. For a role-based example, we might want to add that he played the character whose name was **Dr. Peter Venkman**.

3. **alumni:** attaching 'graduationYear' to an alumni relationship.

Schema.org already has explicit names for both 'alumni' (from an EducationalOrganization to a Person) and its inverse 'alumniOf' (from a Person to an EducationalOrganization). This makes for a convenient example, as we can try out descriptions of this situation from both a Person-centric and from an Organization-centric perspective.

For example we might want to say not only that 'university_of_bristol alumni dan', but that this situation also involved a graduationYear of 1994, or a startDate of 1991.

Vocabulary: introducing Role and hasRole

The schema.org approach here is to introduce a new general-purpose type called "Role" which addresses these and related usecases. This allows us to add additional information as properties of the Role, since it is not possible in the schema.org datamodel to attach annotations directly to the relationship between two entities. Sub-types of Role can optionally be used to help organize role-oriented terminology, but are not strictly necessary. There are potentially many distinct roles that involve any pair (or larger collection) of entities.

In addition to this new Role type hierarchy, we also introduce a property that relates an entity to a specific Role that they are involved in: *hasRole* (feedback is particularly welcomed on this naming choice).

The proposed model for using this new terminology is that any entity-valued property (e.g. actor, alumni, player, …) can now be used in a Role. So in addition to 'actor' being expected on Movie or TVSeries or RadioEpisode, we also can use 'actor' on a **Role**. Similarly, 'alumni' could be used directly on an EducationalOrganization, or it could be used on a Role. The purpose of hasRole is then to link the original entity (e.g. a TVSeries, EducationalOrganization etc.) to this intermediate Role object.

This gives us two patterns. The traditional simple pattern, and a richer variant which uses the intermediate role entity:

In simple text-art form, our 3 examples:

- **SF49ers** --quarterback→ **JoeMontana**.
- **Ghostbusters** --actor→ **BillMurray.**
- **BristolUniversity** --alumni→ DanB.

In each of these cases, we can also represent the relationship by way of attaching the basic property to an instance of a Role, and then attaching that Role to the subject of the relationship."

- SF49ers --hasRole→ [**role7**] --quarterback→ JoeMontana (a Person)
- Ghostbusters --hasRole→ [**role109**] --actor→ BillMurray.
- BristolUniversity --hasRole → [**role261**] --alumni→ DanB.

By introducing these intermediate Role entities, we can then attach further information to them using any schema.org-compatible notation (i.e. Microdata, RDFa, JSON-LD etc.). Common uses will include temporal information (e.g. Joe's quarterback Role with the SF49ers during 1979-1992), alongside also potentially arbitrary other properties and relationships.

Schema:

- <Role> subClassOf <Intangible>.
- <hasRole> a <Property>; domainIncludes <Role>; rangeIncludes <Thing>.
- <MediaRole>, <SportsTeamRole> subClassOf <Role>.

Example 1:

- **JoeMontana**
  - type: Person
  - hasRole: JoeMontanaSF49ersRole01

- **SF49ers**
  - type: FootballTeam
  - hasRole: JoeMontanaSF49ersRole01

- **JoeMontanaSF49ersRole01**
  - type: SportsTeamRole
    - where SportsTeamRole is subClassOf Role
  - player: JoeMontana
  - team: SF49ers
  - startDate: 1979
  - endDate: 1992
  - position: QuarterBack

In the simple pre-role world, we should only have **JoeMontana** and **SF49ers** related directly by a property such as "position". By introducing a 3rd entity, the Role, we have an appropriate attachment point for additional properties, such as the position they played, and the period during which they played it.

How are the 3 entities related? We can use a generic hasRole property to make the 3-way association:

Sport-specific subtyping of Role can alert us to the possible presence of relevant other entities, which can be connected either by their own hasRole, or by domain-specific properties. In the scenario shown here, the role relates to the Team with a 'team' property; and the team relates back to the role with a hasRole property.

Essentially we need a pattern of relationships that associate these 3 entities in a predictable manner.

Note: the Role pattern is not designed to handle every possible situation in which several entities are inter-related. Sometimes a purely case-by-case schema design is appropriate. For example, a http://schema.org/BorrowAction can have an agent Person, a lender Person, an object borrowed etc. While there is no need to apply Role to such situations, there will be many cases where the generic pattern of using a Role is appropriate. It is important to manage expectations here: the Role mechanism is not intended as a pattern that will address all representational needs whenever we have a complex non-binary relationship between several entities. BorrowAction is an example of a case where we have a custom design instead, and there will likely be others.

Markup Examples

How does the Sports example look in concrete markup? Let's try it first with markup that begins with a Team. First we'll start with the unqualified, simple version. Then we'll extend this to add a Role.

Simple flat property: `athlete` described directly via original defined range of `Person`, telling us that someone is an athlete in the team (but leaving lots of other questions unanswered):

```
{
  "@context": "http://schema.org/",
  "@type": "AmericanFootballTeam",
  "name": "San Francisco 49ers",
  "athlete": {
    "@type": "Person",
    "name": "Joe Montana"
  }
}
```

This is straightforward, but doesn't let us say *when* Joe was an athlete for this Team; or specifically which *position* he played. To do that, we need to introduce another entity...

Complex Role pattern:

eg.1-JSON:

```
{
    "@context": "http://schema.org/",
    "@type": "AmericanFootballTeam",
    "name": "San Francisco 49ers",
    "hasRole": {
        "@type": "AmericanFootballRole",
        "@id": "role321",
        "startDate": "1979",
        "endDate": "1992",
        "position": "Quarterback",
        "athlete": {
                "@type": "Person",
                "name": "Joe Montana",
                "hasRole": "role321"
        }
    }
```

```
}
```

Here are the raw triples that a generic JSON-LD parser gives, for this data:

```
<http://json-ld.org/playground/role321> <http://schema.org/endDate>
"1992"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://json-ld.org/playground/role321> <http://schema.org/athlete> _:b1 .
<http://json-ld.org/playground/role321> <http://schema.org/position> "Quarterback" .
<http://json-ld.org/playground/role321> <http://schema.org/startDate>
"1979"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://json-ld.org/playground/role321> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://schema.org/AmericanFootballRole> .
_:b0 <http://schema.org/hasRole> <http://json-ld.org/playground/role321> .
_:b0 <http://schema.org/name> "San Francisco 49ers" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/AmericanFootballTeam> .
_:b1 <http://schema.org/hasRole> "role321" .
_:b1 <http://schema.org/name> "Joe Montana" .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

In this design, we see the *athlete* property applied to the Role, instead of to a Team.

See also more on JSON-LD reverse property syntax. This can be useful for situations where the property is named in an inconvenient direction, e.g. ... for example when describing roles from the perspective of a player in a bio page.

Let's see the other two motivating examples: adding information about an acting role, and alumni.

eg2-JSON:

RECAP: Simple schema.org already allows us to say that **Bill Murray** was a star in the ***Movie*** **Ghostbusters**. For a role-based example, we might want to add that he played the character whose name was **Dr. Peter Venkman**.

Short form:

```
{
   "@context": "http://schema.org/",
   "@type": "Movie",
   "name": "Ghostbusters",
      "actor": {
            "@type": "Person",
            "name": "Bill Murray"
```

```
    }
}
```

Simple triples:

```
_:b0 <http://schema.org/actor> _:b1 .
_:b0 <http://schema.org/name> "Ghostbusters" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Movie> .
_:b1 <http://schema.org/name> "Bill Murray" .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

This can be expanded with an additional intermediate Role entity as follows:

Long form:

```
{
   "@context": "http://schema.org/",
   "@type": "Movie",
   "name": "Ghostbusters",
   "hasRole": {
      "@type": "MovieRole",
      "@id": "movierole_678",
      "characterName": "Dr. Peter Venkman",
      "actor": {
            "@type": "Person",
            "name": "Bill Murray",
            "hasRole": "movierole_678"
      }
   }
}
```

This says "a Movie called 'Ghostbusters' is part of a MovieRole situation, where the actor is a Person called 'Bill Murray'.

Here are the triples:

```
<http://json-ld.org/playground/movierole_678> <http://schema.org/actor> _:b1 .
<http://json-ld.org/playground/movierole_678> <http://schema.org/characterName> "Dr. Peter Venkman" .
<http://json-ld.org/playground/movierole_678> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://schema.org/MovieRole> .
_:b0 <http://schema.org/hasRole> <http://json-ld.org/playground/movierole_678> .
_:b0 <http://schema.org/name> "Ghostbusters" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Movie> .
_:b1 <http://schema.org/hasRole> "movierole_678" .
_:b1 <http://schema.org/name> "Bill Murray" .
```

```
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

eg3a-JSON:

```
{
   "@context": "http://schema.org/",
   "@type": "EducationalOrganization",
   "name": "University of Bristol",
   "hasRole": {
      "@type": "Role",
      "@id": "edurole26151",
      "graduationYear": "1994",
      "alumni": {
            "@type": "Person",
            "name": "Dan Brickley",
            "hasRole": "edurole26151"
      }
   }
}
```

This says: "An EducationalOrganizational 'University of Bristol' is part of a Role situation also involving as an alumni Person, 'Dan Brickley'.

Triples:

```
<http://json-ld.org/playground/edurole26151> <http://schema.org/alumni> _:b1 .
<http://json-ld.org/playground/edurole26151> <http://schema.org/graduationYear> "1994" .
<http://json-ld.org/playground/edurole26151> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://schema.org/Role> .
_:b0 <http://schema.org/hasRole> <http://json-ld.org/playground/edurole26151> .
_:b0 <http://schema.org/name> "University of Bristol" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/EducationalOrganization> .
_:b1 <http://schema.org/hasRole> "edurole26151" .
_:b1 <http://schema.org/name> "Dan Brickley" .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

Final example: reverse properties. How would all of this work if our property was named in an inconvenient direction? We can show this using 'alumni' since schema.org also has an explicitly named inverse property, 'alumniOf'. And as mentioned above, JSON-LD has @reverse keyword, which allows us to express a relationship that is named pointing to rather than from some entity.

RDFa also has @rev syntax, and discussion is underway about adding something equivalent into Microdata.

So let's describe the exact same situation but imagine we have to use alumniOf instead of alumni, to make sure that the underlying pattern can still be represented:

eg3b.

```
{
   "@context": "http://schema.org/",
   "@type": "EducationalOrganization",
   "name": "University of Bristol",
   "hasRole": {
      "@type": "Role",
      "@id": "edurole321",
      "graduationYear": "1994",
      "@reverse": {
        "alumniOf": {
            "@type": "Person",
            "name": "Dan Brickley",
            "hasRole": "edurole321"
        }
      }
   }
}
```

Raw triples:

```
<http://json-ld.org/playground/edurole321> <http://schema.org/graduationYear> "1994" .
<http://json-ld.org/playground/edurole321> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://schema.org/Role> .
_:b0 <http://schema.org/hasRole> <http://json-ld.org/playground/edurole321> .
_:b0 <http://schema.org/name> "University of Bristol" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/EducationalOrganization> .
_:b1 <http://schema.org/alumniOf> <http://json-ld.org/playground/edurole321> .
_:b1 <http://schema.org/hasRole> "edurole321" .
_:b1 <http://schema.org/name> "Dan Brickley" .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

Closing Notes

What does this mean for people developing and using schema.org schemas?

Firstly, this design is mostly for users/publishers. It is a work-around pattern that deals with the

constraints inherent in the simple graph data-model we use (and that underpins Microdata, RDFa, JSON-LD). It provides publishers a useful general purpose type - "http://schema.org/Role" - that can be conveniently used across the entire range of schema.org descriptive scenarios. Since schema.org now has 500+ types, this is useful. However it is not the only option, and in some cases we will find it appropriate to define similarly structured but more task-specific representations (such as BorrowAction).

This design in effect adds "Role" as a possible type for *every entity-valued schema.org property.* In other words, every schema.org property that relates two entities (e.g. menu, actor, event, ...) could be used on a Role instead, alongside a 'hasRole' property which relates any kind of thing to the intermediate role. We can conceptualize schema.org's normal entity-valued properties as linking a "*from-entity*" to a "*to-entity*" (e.g. 'actor' might have from-entity of Ghostbusters, to-entity Bill Murray) (in RDF terms, these are sometimes called 'subject' and 'object' respectively). In these terms, a Role entity serves to substitute for the 'from-entity', i.e. the 'actor' relationship will relate a role to "to-entity' Bill Murray.

APPENDIX

JSON-LD @context practicalities.

Currently schema.org does not yet publish a "@context file". This is a JSON-LD document that aids JSON-LD parsers to interpret property values. In the absence of such a file, the 'hasRole' references in the above examples may not parse correctly into entity references. A workaround for this is to include inline @context property definitions for hasRole, to make sure parsers understand that its values are entity references not strings.

For example,

```
{
   "@context": "http://schema.org/",
   "@type": "Movie",
   "name": "Ghostbusters",
   "hasRole": {
      "@type": "MovieRole",
      "@id": "movierole_678",
      "characterName": "Dr. Peter Venkman",
      "actor": {
            "@type": "Person",
            "@context": {"hasRole":{"@type":"@id"}},
            "name": "Bill Murray",
```

```
            "hasRole": "movierole_678"
        }
    }
}
```

Alternate designs.

The issues addressed by the mechanism described above are widely encountered when working with an RDF-like graph data model.

Alternatives that have been explored elsewhere include:

1. Extend the underlying data model to allow per-property annotations. The 'property graph' model attempts this (see <https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model>). For some simple literal-valued property annotations this can be a useful approach. However it is a different data model to that used by schema.org (and by Microdata, JSON-LD and RDFa in general).

2. "RDF Reification". The original 1997 RDF Model and Syntax specification <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/> included a small vocabulary for describing ("reifying") entire RDF graphs by encoding them using a "Statement" type and 'subject', 'predicate' and 'object' properties. It was a more ambitious approach than that presented here, and had different goals. In this classic RDF reification approach, reified statements were explicitly not considered to be asserted. By contrast, information encoded via Role is considered to be a description of the entity described (although there are some subtleties around temporal qualifications here).

3. Other n-ary patterns. There are a variety of modeling conventions for representing relationships between more than two entities. A W3C Note describes some of these, <http://www.w3.org/TR/swbp-n-aryRelations/>. The notion of a Compound Value Type (CVT) in Freebase is also related <http://wiki.freebase.com/wiki/Compound_Value_Type>.

4. Using two properties to associate entities with the Role. Schema.org welcomes feedback on the idea that we could instead use two different property names to related a Role to its associated entities. The example from above expressed in this model would look instead like this:

```
{
    "@context": "http://schema.org/",
    "@type": "Movie",
    "name": "Ghostbusters",
```

```
    "hasRole": {
        "@type": "MovieRole",
        "@id": "movierole_678",
        "characterName": "Dr. Peter Venkman",
        "actor": {
                "@type": "Person",
                "name": "Bill Murray",
                "inRole": "movierole_678"
        }
    }
}
```

Here we write 'inRole' to relate BillMurray back to our MovieRole instance. Although introducing an additional property into the design can be seen as complicating, it may also help clarify the respective contributions played by Ghostbusters and by BillMurray regarding the Role. We welcome discussion on this design option, as well as on the choice of property names. For example, would 'playsRole', 'containsRole' or similar work better here?