Actions in schema.org

draft 4

Overview

This document proposes the introduction of actions to the schema.org vocabulary, in the form of a new Class of Things, called Actions.

Actions are used to describe:

(a) an ability/intent to perform a certain operation (proposed) or
(b) an operation that already happened (completed).

Each action has corresponding arguments/slots/parameters that are well defined.

Actions define a standard programmatic pre-defined interface between parties (e.g. which arguments "Watching a Movie" takes), and ActionHandlers helps with the mechanisms

(e.g. invoking an action via an android intent vs a HTTP GET).

Actions allows the machine-readable brokering language between intent providers, registries and intent consumers.

Goals and Non-Goals

We wanted to:

- Model past, present and future actions.
- Model invocation of actions.
- Define a list of well known actions and contextual arguments.
- Define an interface that developers can implement.
- Be syntax and mechanism agnostic: we want this to work on multiple platforms (e.g. http/html, smtp/pop/mime, http/rest, http/feeds and mobile apps).

We explicitly do not want to:

- Allow ambiguous or poorly specified verbs.
- Allow for arbitrary interface modelling (e.g. allowing the dynamic expression of required/optional properties).
- Allow for a complex discovery and registration mechanism for unimportant use cases.
- Model propositional attitudes or beliefs.

Design

This section goes over a few design decisions we took while modelling actions.

Generalization

We wanted the ability to refer/process actions generically, for consistency and convenience. For example, since http://schema.org/VideoObject inherits from http://schema.org/CreativeWork and ultimately http://schema.org/Thing, a data consumer can fallback and walk up the tree and stop at whichever level of specificity it chooses to understand.

In that context, we wanted all actions to be cast-able to a common base ancestor and be treated generically if need be. We chose to call this base class a http://schema.org/Action and we build a hierarchical tree below it.

Specialization and disambiguation

At the same time we wanted to facilitate the consumption of actions, we wanted to allow specialization too.

The single most important problem we wanted to solve was disambiguation. We wanted to make sure that there was a *very* clear mapping between the user intent and which verb to pick. For example, we really wanted to avoid perfect-match synonyms (2 verbs that mean the exact same thing, like “buy” and “purchase”) although we allow synonyms that are generalizations of another (like “trade” is a generalization of “buy”).

We allow reciprocals (e.g. Buy and Sell) and antonyms (e.g. Accept and Reject).

To address these problems, we propose a taxonomy of verbs organized in a tree. The general idea of the tree is that paths are made of synsets and as you walk down the tree you get to more specialized verbs of the more general concept (e.g. “appending” is “inserting” at the end).

We use framenet, wordnet and webster to inform the construction of our tree.

Parametrization

Each verb has a specific set of contextual parameters (or arguments or slots or semantic roles or properties, depending on who you are asking).

We use case grammar, more specifically framenet and wordnet, to inform which verbs take which semantic roles. Here is one example of which roles applies to Buy according to framenet.

Arg0-pag: *buyer* (vnrole: 13.5.1-agent)
Arg1-ppt: *thing bought* (vnrole: 13.5.1-theme)
Arg2-dir: *seller* (vnrole: 13.5.1-source)
Arg3-vsp: *price paid* (vnrole: 13.5.1-asset)
Arg4-gol: *benefactive* (vnrole: 13.5.1-beneficiary)

Since verbs are laid on a tree, more specific verbs inherit the parameters from their more generic ancestors.

Temporalization

Actions can be found in different temporal states (e.g. verb tenses, past, present, future, imperative forms). Each action instance is in one of well known states (proposed, pending, completed or cancelled).

We considered using named graphs, but we got the feedback from real world developers that that was less readable and overly complex.

We settled on creating an ActionStatus enumeration of the possible states an action can be at.

Invocation

We created an invocation mechanism that allows machines to programatically execute actions.

Proposed actions are invoked via ActionHandlers, which describe the mechanisms to invoke the action. There is a base ActionHandler type, and per-platform-technology sub-types, like WebPageHandler, HttpHandler and MobileHandler.

Registration & Discovery

Discovery happens via crawling exposed actions. Developers/publishers expose what they can do and crawlers reach out to find that. There isn't a global registry where service providers can go to to register, all actions are found in the same context the items they apply to are.

Put simply, there are two ways that you can express what you can do: (1) Entities to Actions and (2) Actions to Entities.

On the former, you add your actions to your existing schema.org entities via the Thing.operation property, which refers to the actions that can be taken on a specific item. Here is one example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Movie"
  "name": "Like Crazy"
  "url": "http://movies.netflix.com/WiMovie/Like_Crazy/70167118",
  "operation": {
    "@type": "WatchAction"
    "status": "proposed",
    "handler" : {
      "@type": "WebPageHandler",
      "url":
"http://movies.netflix.com/WiPlayer?movieid=70167118&t=Like+Crazy",
    }
    "handler" : {
      "@type": "AndroidHandler",
      "url":
"http://movies.netflix.com/WiPlayer?movieid=70167118&t=Like+Crazy",
      "package": "com.netflix",
    }
  }
}
</script>
```

On the former, you expose an Action and the entities it applies to. The actual mechanism/syntax to allow this is still being discussed.

Specification

The following section describes the specification of each of the concepts mentioned before.

Thing

Property	Type	Description
operation	Action	An operation that can be taken on the item.

Example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "MusicGroup"
  "name": "Van Halen"
  "url": "http://www.pandora.com/van-halen",
  "operation": {
    "@type": "ListenAction"
    "actionStatus": "PROPOSED",
    "actionHandler" : {
      "@type": "WebPageHandler",
      "url": "http://www.pandora.com/van-halen",
    }
  }
}
</script>
```

Thing > Action

Action extends Thing.

Property	Type	Description
agent	Organization or Person	The direct performer or driver of the action (animate or inanimate). e.g. *John* wrote a book.
object	Thing	The object upon the action is carried out, whose state is kept intact or changed. Also known as the semantic roles patient, affected or undergoer (which change thei

		state) or theme (which doesn't). e.g. John read *a book*.
participant	Organization or Person	Other co-agents that participated in the action indirectly. e.g. John wrote a book with *Steve*.
result	Thing	The result produced in the action. e.g. John wrote *a book*.
location	Place or PostalAddress	The location of the event, organization or action.
instrument	Thing	The object that helped the agent perform the action. e.g. John wrote a book with *a pen*.
actionHandler	ActionHandler	The handler that allows the invocation of this action.
actionStatus	ActionStatus	The status of the action.
startTime	DateTime	When the Action was performed: start time. This is for actions that span a period of time. e.g. John wrote a book from *January* to December.
endTime	DateTime	When the Action was performed: end time. This is for actions that span a period of time. e.g. John wrote a book from January to *December*.

Example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "BuyAction"
  "agent": {
    "@type": "Person"
    "name": "John",
  }
  "object": {
    "@type": "Product"
    "name": "iPod",
  }
  "actionStatus": "COMPLETED"
}
</script>
```

Thing > Intangible > Enumeration > ActionStatus

Proposed	The action is proposed.
Pending	Action is in the process of being executed.
Completed	Action has been executed.
Cancelled	Action was requested but was cancelled

Thing > Property > SupportedProperty

A property known to be supported by a class.

Property	Type	Description
required	boolean	Whether the property is required or not.

Thing > Intangible > ActionHandler

A mechanism to invoke and fulfill an action.

Property	Type	Description
expects	SupportedProperty	Property that can be specified on the Action that the handler supports.
returns	Class	The type of the instance returned in this call.

Example (reference):

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Please rate your experience.",
  "operation": {
    "@type": "ReviewAction",
    "review": {
      "@type": "Review",
      "itemReviewed": {
        "@type": "FoodEstablishment",
```

```
        "name": "Joe's Diner"
      },
      "reviewRating": {
        "@type": "Rating",
        "bestRating": "5",
        "worstRating": "1"
      }
    },
    "actionHandler": {
      "@type": "HttpActionHandler",
      "url": "http://reviews.com/review?id=123",
      "expects": {
        "@type": "SupportedProperty",
        "name": "review.reviewRating.ratingValue",
        "required": "true",
      },
      "method": "POST"
    }
  }
}
</script>
```

Thing > Intangible > ActionHandler > WebPageHandler

A handler that fulfills the actions by taking the user to a web page.

Property	Type	Description
url	URL	The deep link where the action can be completed.

Example (reference)

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Watch the 'Avengers' movie online"
  "operation": {
    "@type": "ViewAction",
    "actionHandler": {
      "@type": "WebPageHandler",
      "url": "https://watch-movies.com/watch?movieId=abc123"
    }
  },
}
</script>
```

Thing > Intangible > ActionHandler > WebPageHandler > WebParamsHandler

Property	Type	Description
httpMethod	Text	GET or POST.

Thing > Intangible > ActionHandler > HttpHandler

A handler that fulfills the action by making a HTTP call to an external API.

Property	Type	Description
url	URL	The deep link where the action can be completed.
httpMethod	Text	POST or GET.

Example (reference):

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Approval request for John's $10.13 expense",
  "operation": {
    "@type": "ConfirmAction",
    "name": "Approve Expense",
    "actionHandler": {
      "@type": "HttpHandler",
      "url": "https://myexpenses.com/approve?expenseId=abc123"
    }
  }
}
</script>
```

Thing > Intangible > ActionHandler > AndroidHandler

A handler that fulfills the action by dispatching to an android intent.

Property	Type	Description

packageName	Text	The name of the package the intent needs to be fired to.

Example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Movie",
  "url": "http://movies.netflix.com/Movie/The_Matrix/20557937",
  "operation": {
    "@type": "WatchAction",
    "actionHandler": {
      "@type": "AndroidHandler",
      "url": "com.netflix.Netflix"
    }
  }
}
</script>
```

Thing > Intangible > ActionHandler > WebFormHandler

A handler that fulfills the action by submitting a specific form on the web. User agents are expected to either (a) direct the user to the HTML form or (b) gather the HTML form elements inside the scope of the handler and use them to build UI for users (taking the right precautions with regards to security - e.g. XSS - and privacy - e.g. submission tokens).

This handler is based on the Yandex specification file.

Property	Type	Description

Example:

```
<div itemscope itemtype="http://schema.org/Movie">
  <span itemprop="name">Transformers</span>
  <div itemprop="operation" itemscope
itemtype="http://schema.org/ReviewAction">
    <form id="theform" action="http://example.org/foobar" method="POST"
itemprop="actionHandler" itemscope
itemtype="http://schema.org/WebFormHandler"  >
      <input type="textarea" name="comments">
      <input type="submit" name="write review">
```

```
    </form>
  </div>
</div>
```

Thing > Action > *

- Action: agent, endTime, instrument, location, object, participant, result, startTime
    - AchieveAction
        - LoseAction: winner
        - TieAction
        - WinAction: loser
    - AssessAction
        - ChooseAction: option
            - VoteAction: candidate
        - IgnoreAction
        - ReactAction
            - AgreeAction
            - DisagreeAction
            - DislikeAction
            - EndorseAction: endorsee
            - LikeAction
            - WantAction
        - ReviewAction: resultReview
    - ConsumeAction
        - DrinkAction
        - EatAction
        - InstallAction
        - ListenAction
        - ReadAction
        - UseAction
            - WearAction
        - ViewAction
        - WatchAction
    - CreateAction
        - CookAction: foodEstablishment, foodEvent, recipe
        - DrawAction
        - FilmAction
        - PaintAction
        - PhotographAction
        - WriteAction: language

- FindAction
  - CheckAction
  - DiscoverAction
  - TrackAction: deliveryMethod
- InteractAction
  - BefriendAction
  - CommunicateAction: about, language, recipient
    - AskAction: question
    - CheckInAction
    - CheckOutAction
    - CommentAction
    - InformAction: event
      - ConfirmAction
      - RsvpAction
    - InviteAction: event
    - ReplyAction
    - ShareAction
  - FollowAction: followee
  - JoinAction: event
  - LeaveAction: event
  - MarryAction
  - RegisterAction
  - SubscribeAction
  - UnRegisterAction
- MoveAction: fromLocation, toLocation
  - ArriveAction
  - DepartAction
  - TravelAction: distance
- OrganizeAction
  - AllocateAction: purpose
    - AcceptAction
    - AssignAction
    - AuthorizeAction: recipient
    - RejectAction
  - ApplyAction
  - BookmarkAction
  - PlanAction: scheduledTime
    - CancelAction
    - ReserveAction: scheduledTime
    - ScheduleAction
- PlayAction: audience, event
  - ExerciseAction: course, diet, distance, exercisePlan, exerciseType, fromLocation, oponent, sportsActivityLocation, sportsEvent, sportsTeam,

toLocation
        - PerformAction: entertainmentBusiness
- SearchAction: query
- TradeAction: price
    - BuyAction: vendor, warrantyPromise
    - DonateAction: recipient
    - OrderAction
    - PayAction: purpose, recipient
    - QuoteAction
    - RentAction: landlord, realEstateAgent
    - SellAction: buyer, warrantyPromise
    - TipAction: recipient
- TransferAction: fromLocation, toLocation
    - BorrowAction: lender
    - DownloadAction
    - GiveAction: recipient
    - LendAction: borrower
    - ReceiveAction: deliveryMethod, sender
    - ReturnAction: recipient
    - SendAction: deliveryMethod, recipient
    - TakeAction
- UpdateAction: collection
    - AddAction
        - InsertAction: toLocation
            - AppendAction
            - PrependAction
    - DeleteAction
    - ReplaceAction: replacee, replacer