Schema.org Actions (Minimal draft)

Status: a minimal draft of an Actions mechanism, based on schema.org partner discussions.

Feedback is welcomed via the W3C WebSchemas group (<public-vocabs@w3.org>)

Overview

This document provides a minimalistic overview of an “Action” mechanism for schema.org.

Schema.org currently defines many named types for classes/categories of thing (CreativeWork, Movie, Person etc.), alongside various named properties associated with these types. This specification introduces a new basic type, “Action”, to facilitate the description of potential, current and past actions associated with these various kinds of thing. We focus on describing specific actions, while also providing some machinery to facilitate generalisation.

Here we describe the basic vocabulary structures proposed, to allow actions to be described, associated with relevant entities, as well as handled by systems that can carry out those actions in the world. Some pieces are described as ‘optional’ (mostly around meta-classes); for now these are left in, to provide a complete picture of the proposed machinery.

Scope

This document describes a schema for defining actions and action handling. It should be useful using any concrete representation of schema.org data (e.g. RDFa, Microdata).

The primary motivation are to have ways to semantically describe:

1. Actions that have occurred. Examples:
    a. User A **rented** movie B.
    b. User A **rated** restaurant C.
    c. User A **accepted** invitation D.
2. Describing potential actions and instructions for user agents to perform the action. Examples:
    a. Movie B can be **rented** by visiting http://mysite.com/xyz.
    b. Restaurant C can be **rated** with the RestaurantsReviews mobile application.
    c. Invitation D can be **accepted** by posting a protocol message to url http://mysite.com/cgi
3. Performing an action. Examples:
    a. User Agent asks system X to **rent** movie B for user A.

b. User Agent asks system X to **rate** restaurant C for user A with rating Y.
c. User Agent asks system X to **accept** invitation D for user A.

The proposal includes these core ideas:

1. **Action** - a base schema for all actions, allowing task-specific sub-types such as BuyMovieTicket
2. An initial list of useful **Action** sub-types.
3. **ActionHandler** - a base schema for specifying how an action can be performed.
4. Two useful Action Handler sub-types: **WebActionHandler**, **HttpActionHandler**.
5. Platform-Specific Action Handlers (not covered by the proposal).

Basic Action-related vocabulary

The new vocabulary described here, once agreed, will be represented in HTML+RDFa+RDFS for integration into the schema.org site documentation system.

- The type "**Action**" represents "An action that may be carried out by some 'performer' such as a Person or Organization."
- "Action" is a sibling type to "**Event**", sharing some of its properties, such as 'performer' and spatial/temporal information.
    - Applicable properties include: 'performer', 'duration', 'startDate', 'endDate' …
    - **Action** can be used to describe prior activities, including but not limited to those whose performance occurred via the mechanisms described here.
    - **Action** can also describe actions which are only potential, or which are currently occurring, or which were in progress but were cancelled.
- We introduce an **'actionStatus'** property, which relates an Action to one of several enumerated **ActionStatusValue** options: **PossibleAction ('The action has not been performed yet')**, **PendingAction** ("The action is in the process of being performed."), **CompletedAction** ("The action was performed"), **CancelledAction** ("The action was cancelled.").
- We introduce an **ActionStatusValue** type (an **Enumeration**), as a container for this enumeration.
- We introduce a **possibleAction** property ("An action that might be performed with this Thing."). The value of a **possibleAction** property is expected to be an **Action**. The property applies directly to any particular **Thing** that the action is applicable to.
    - for example, the Movie *Skyfall* might have...
        - **possibleAction**: _act_1_watch_online_5121 (a '**RentMovieOnline'** Action)
        - **possibleAction**: _act_2_buy_ticket_2511 (a '**BuyMovieTicket'** Action)
            - *these Action instances represent very specific (geography, person, delivery, …) action opportunities, and might be described with additional properties.*

        - **RentMovieOnline** and **BuyMovieTicket are sub-types of Action (**and *optionally*, also of type **ActionType**).
- *Optionally*, we introduce a parallel property "**eventStatus**", Enumeration type, **EventStatusValue**, and enumerate instances including (per prior proposal) at least **CancelledEvent**, **RescheduledEvent, TentativeEvent, ConfirmedEvent** (… others?)
- *Optionally*, we introduce a type '**Class**', defined as equivalent to rdfs:Class, a type "Property", defined as equivalent to rdf:Property,
- ...and a sub-type of this new '**Class'** type named "**ActionType**', "Types used for defining actionable subclasses of Action.".
    - ActionType can be used to indicate specific sub-types of Action which are designed to participate in the Action-description machinery described here. While not essential, this allows other useful sub-types of Action to be defined for other purposes without confusion.
- *Optionally, w*e introduce a **possibleActionType** property ("A (sub)type of Action that might be performed with this type of Thing."). The value of a **possibleActionType** property is expected to be an **ActionType**, and the property is expected to be applied to a **Class**.
    - for example, the class **Movie** might have:
        - **possibleActionType**: **RentMovieOnline**
        - **possibleActionType**: **BuyMovieTicket**
            - *these generalisations serve as hints; often it is more useful to describe a specific action opportunity as a detailed 'one-off' instance, rather than trying to infer the possibility from general rules.*

ActionHandlers

So far, we have only described the world in more detail. By introducing the ActionHandler mechanism, we introduce enough details for a user-agent to actually carry out the action on behalf of some party.

- We introduce an **Intangible** sub-type, "**ActionHandler**", for describing how Actions can be implemented - "An application or service that allows an Action to be performed."
- User Agents may invoke **ActionHandler**s in order to perform an action. An ActionHandler is typically passed a description of the Action instance as input.
- ActionHandlers may specify what properties of the Action need to be completed before the Action can be handed to them.
- We introduce a property **'handler**' that associates a specific Action instance with a specific **ActionHandler**.
- We introduce a property **'optionalProperty'** that applies to an **Action**, and which expects values that are a **Property.** "Property that can be specified on the Action that the handler supports."

- We introduce a property **‘requiredProperty’** that applies to an **Action**, and which expects values that are a **Property.** “Property that must be given in an Action description, for it to be handled by its ActionHandler.”
- We introduce a property ‘**actionType**’ that applies to an ActionHandler, and which expects values that are a **ActionType**. This indicates the general type(s) of action that some handler can accept.
- We introduce the following sub-types of **ActionHandler:**
    - WebActionHandler, “Any action which can be carried out by simply following an **url** (property of the **Action**) is handled by a **WebActionHandler**”.
    - **HTTPActionHandler,** with properties ‘url’ (inherited from Thing), ‘method’, **‘encoding’**, **‘authorization’**
        - We introduce a type **‘HTTPRequestMethod’** as a sub-type of **Enumeration**. We enumerate two instances of it: **HTTPGET**, **HTTPPOST**.
        - TODO: Examples/detail for **‘encoding’**, **‘authorization’.** E.g. ‘http://schema.org/Encoding/JSON’. More?
    - We anticipate additional protocol and platform-specific handlers in the future.

Specific Action Schemas

In addition to all this general machinery, we introduce some more specific utility vocabulary.

First, several specific Action Schemas:

- **AcceptAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for confirming something, e.g. an invitation, an offer.”
    - new property **‘accepted’** (“The thing that is accepted.”), expected on **AcceptAction**, expected value: **Thing**
- **AddAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for adding something (e.g. a product to a shopping cart, a friend to a friend list).”
    - new property ‘**added**’ (“The thing that is added.”), expected on AddAction, expected value: **Thing**
- **BuyAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for buying something (e.g. a product).”
    - new property ‘**bought**’ (“The thing that is bought.”), expected on BuyAction, expected value: **Thing**
- **CancelAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for canceling something (E.g. a reservation, a purchase, an event).”
    - new property ‘cancelled’ (“The thing that is cancelled.”), expected on CancelAction, expected value: **Thing**
- **CheckInAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for checking-in (e.g. a flight, a location).”

    - new property “checkedInTo” (“The thing that is checked into.”), expected on CheckInAction, expected value: **Thing**
- **CommentAction**, a sub-type of **Action** (and of type **ActionType**)
    - “Action for leaving a comment (e.g. on an article).”
    - new property “checkedInTo” (“The comment that is commented.”), expected on CheckInAction, expected value: **Comment**
        - See ISSUE re http://schema.org/UserComments
- [several more omitted for now...]

Open Issues

- **ISSUE**: **Action identity conditions across lifecycle.**
    - ActionHandler (alongside other out-of-band processes) allow actions to progress through a lifecycle. While the status possible/pending/completed/cancelled can evolve, should we create new entities at each step, and link them? Or treat it as one entity with changing characteristics?
- **ISSUE: UserComments cleanup.** We already added Comments class as partial cleanup for http://schema.org/UserComments … can more be done now?
- Should we get the **Property** and **Class** types defined in advance of this. Conclusion: a separate proposal has been published, http://www.w3.org/wiki/WebSchemas/SchemaDotOrgMetaSchema
- Vocabulary choice: let’s try to avoid types and properties differing only by case; what better name can we have for the type of values for the property ‘**expectedActionType’?**

TODOs

The following are areas for further consideration and work:

- An **Enumeration** sub-type, **Authorization;** with instances**: HTTP BasicAuthentication; OAuth2Authentication.**
- Chaining of action handlers.
- Translation of all examples into new structures.
- Illustrations / figures.

Requirements

1. It should be possible to use Schema.org Actions to describe very specific ‘possible action’ opportunities, without requiring those potential actions to be determined by inspecting general type-level descriptions. For example, the potential actions for Movies

vary between movies, delivery environment, geography etc.
2. It should be possible (but not required) to express generalizations about types of actions and types of Thing they apply to.
3. It should be possible (but not required) to use additional services and knowledge to determine additional relevant Action instances and sub-types.
4. Support but don't initially enumerate Operating System -specific subtypes of **ActionHandler**.

New Vocabulary

To summarize the new vocabulary presented here.

Related core-vocabulary:

- Class and Property will be added to schema.org, as aliases for rdfs:Class, rdf:Property.

Essential new vocabulary:

- **Action**, a **Class**
  - **possibleAction**, a Property
  - **actionStatus**, a Property
    - **ActionStatusValue**, a Class.
    - **PossibleAction**, **PendingAction**, **CompletedAction**, **CancelledAction**; all **ActionStatusValue** instances.
  - **actionType**, a Property
    - In English we can say its values are types/classes
    - When **Class** is added (as agreed Nov 29) we can say it expects **Class** values.
    - If **ActionType** is added, we can say it expects those.
    - ISSUE: Can we try to avoid names of types and properties differing only in case? e.g. **expectedActionType** ?
- **ActionHandler,** an Intangible
  - with instances, **WebActionHandler** and **HTTPActionHandler**;
  - **method**, **encoding**, **authorization** properties of **HTTPActionHandler**.

Would-be-nice new vocabulary:

- ActionType, a Class that is a sub-class of Class.
  - see http://www.w3.org/wiki/WebSchemas/SchemaDotOrgMetaSchema

Supporting vocabulary:

- A set of 'specific action schemas'; at this stage illustrative only.

Examples

Here is a simple example of instance and class level representations of potential actions.

In English: We have a Movie, Skyfall. The user can RentMovieOnline either in StandardDefinition from Amazon or StandardDefinition / HighDefinition, from iTunes. The user can also BuyTicketForMovie from Fandango or from TicketMaster. The key here is the use of actionStatus with a value of PossibleAction. For this example we use an informal attribute/value notation:

Skyfall (a Movie)
possibleAction: RMO1, RMO2, RMO3, BTM1, BTM2

Movie (a Class)
possibleActionType: BuyTicketForMovie, RentMovieOnline

RMO1 (a RentMovieOnline)
resolution: StandardDefinition
provider: Amazon
itemRented: Skyfall
actionStatus: PossibleAction

RMO2 (a RentMovieOnline)
resolution: StandardDefinition
provider: iTunes
itemRented: Skyfall
actionStatus: PossibleAction

RMO3 (a RentMovieOnline)
resolution: HighDefinition
provider: iTunes
itemRented: Skyfall
requiredSoftware: ...
actionStatus: PossibleAction

BTM1 (a BuyTicketForMovie)
provider: Fandango
movie: Skyfall
actionStatus: PossibleAction

BTM2 (a BuyTicketForMovie)
provider: TicketMaster

movie: Skyfall
actionStatus: PossibleAction

Explanation

Consider the first example Action, **RMO1**. Here the specific (possible) action is an instance of the type “RentMovieOnline”, which itself is a sub-type of schema.org/Action. We may decide to also declare this type as being an ActionType, but that is not essential here.

There are therefore two ways here that we can represent that this action applies to the Movie Skyfall:

1. The ‘possibleAction’ property is issued directly to state that Skyfall has a possibleAction relationship to the action RMO1.
2. The ‘possibleActionType’ property is issued to associate the general type Movie with some relevant sub-types of Action, namely BuyTicketForMovie, RentMovieOnline. Since RMO1 is of type RentMovieOnline, and Skyfall is of type Movie, we can conclude that the action *may* be applicable to this Movie.

This combination of mechanisms allows for fine-grained accuracy via specific per-action descriptions, as well as leaving open the possibility for type-level generalisation.