Actions in schema.org

draft 5

- Overview
- Use Cases
- Goals and Non-Goals
- Design
    - Registration
        - Individual Resources
        - Collections
    - Invocation
    - Parametrization
    - Temporalization
    - Generalization
    - Specialization and disambiguation
    - Protocol
        - Discovery
        - Serialization
- Specification
    - Basic concepts
        - Thing
        - Action
        - ActionStatus
        - ActionHandler
            - WebPageHandler
            - ApplicationHandler
                - WindowsActionHandler
                - WindowsPhoneActionHandler
                - AndroidHandler
    - Advanced concepts
        - HttpHandler
        - SupportedProperty
        - SupportedClass
        - WebParamsHandler
        - WebFormHandler
    - Action Hierarchy

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) **operations** that **can be taken** on a specific **resource** (proposed) or
(b) **operations** 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 vs iOS apps vs Native Windows applications).

Use Cases

There is a wide range of operations we need to model and throughout this doc we used the following use cases to inform our design:

- **Movies** that can be **watched**, **songs** that can be **listened** (e.g. netflix, spotify)
- **Products** that can be **reviewed** (e.g. imdb, amazon, yelp)
- **Events** that can be **RSVPed** (e.g. eventbrite)
- **Reservations** that can be **confirmed**/**cancelled** (e.g. yellow cab)
- **Expense approvals** that can be **confirmed**
- **Flights** that can be **reserved** or **checked-in** (e.g. aa.com)
- **Cars** that can be **rented** (e.g. hertz, avis)
- **Packages** deliveries that can be **tracked** (e.g. ups)
- **Taxis** that can be **reserved** (e.g. yellow cab)
- **Offers** that can be **saved** (e.g. groupon)
- **Restaurants** that allow **reservations** (e.g. opentable)
- **Restaurants** that allow **orders** for delivery or pickup (e.g. grubhub)
- **Hotels** that can **reserve** rooms (e.g. hilton)
- **Airlines** that can **find** flights (e.g. aa.com)
- **Local Businesses** that can **schedule** appointments (e.g. yelp)
- **Organizations** that allow you to **search** for Stores (e.g. walmart)

We created a Demo of these use cases to help us understand what these look like in real life.

Goals and Non-Goals

We want 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 transport 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 description of arbitrary dynamic interfaces with arbitrary parameters and return values).
- Model propositional attitudes or beliefs.

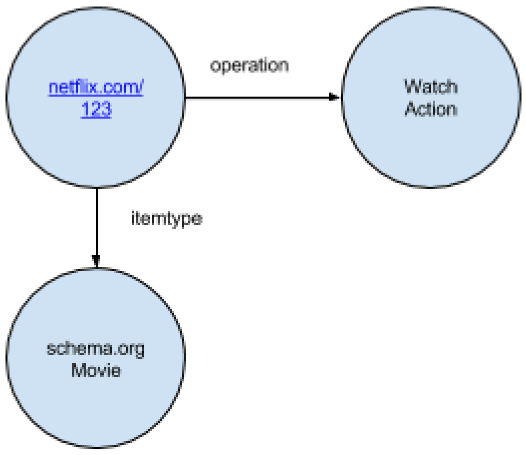
Design

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

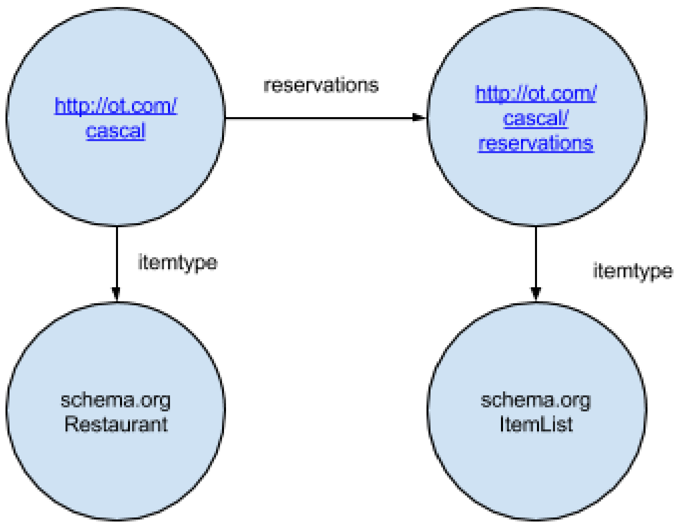
Registration

Put simply, actions are taken **on the resource** where the **operation** is attached to.

It follows the RESTful approach in the sense that it deals with resources identified by URIs and operations performed on them. It differs from REST in the sense that it provides a much wider set of verbs (e.g. O(100)s rather O(4)) and a much tighter set of parameters/properties that can be used.



It follows the Linked Data approach in the sense that resources can point to other resources (which in turn can point to more resources) via semantically meaningful links.



Having that in mind, there are a couple of use cases that are important to note: individual resources and collection resources.

Individual Resources

Individual resources are the simplest form of resources: they refer to an individual entity where operations can be attached. 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 that specific item/resource. Here is an example of a movie that can be watched:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Movie"
  "@id": "http://movies.netflix.com/WiMovie/70167118",
  "name": "Like Crazy"
  "operation": {
    "@type": "WatchAction"
    "url": "http://movies.netflix.com/WiPlayer/70167118/watch",
    “actionStatus”: “proposed”,
  }
}
</script>
```

Collections

Some resources are containers of resources (e.g. a http://schema.org/ItemList). In those cases, actions are still taken on a resource (since an ItemList is a type as valid as any other type), but it gives you the ability to bind operations to a set of resources rather than individuals.

Since the collection resource is bound via a named property (which has its own description as well as typing), there is a meaningful contract on the expectations of what the resource will contain.

For example, lets say you wanted to find a table in a restaurant. The restaurant could expose their reservations via a collection of reservations which can be searched on.

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Restaurant"
  "@id": "http://opentable.com/cascal"
  "name": "Cascal"
  "reservations": {
    "@type": "ItemList"
    "@id": "http://opentable.com/cascal/reservations"
    "operation": {
      "@type": "SearchAction"
      "url": "http://www.opentable.com/cascal-reservations-mountain-view"
      “actionStatus”: “proposed”
    }
  }
}
</script>
```

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 AndroidHandler.

In the spirit of keeping simple things simple, an url attached to an Action is semantically equivalent to an Action with a WebPageHandler attached to it with that url.

We'll go over in more detail how each of those work below on their specification.

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.

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/specializations 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.

Protocol

It is really important to note that this document only refers to the vocabulary of actions (i.e. the words you use to describe the actions), as opposed to things like transport mechanisms, authentication and authorization, caching expectations and security tokens.

These are things that are still evolving and we’ll only be able to standardize once we have enough implementations.

Here are some of the major areas that we are still exploring:

Discovery

These are some of the discovery mechanisms that we are exploring as part of the protocol design.

On the web, discovery can happen via crawling exposed actions in formats such as markup, JSON-LD or sitemaps. Developers/publishers expose what they can do and crawlers reach out to find that.

In the world of application platforms, apps could register via a manifest, for example, and then discovery would occur via the platform's preferred registry system (e.g. android intents registry, windows registry, etc).

Specification

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

Basic concepts

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",
  "@id": "http://www.pandora.com/van-halen",
  "@type": "MusicGroup"
  "name": "Van Halen"
  "operation": {
    "@type": "ListenAction"
    “actionStatus”: “PROPOSED”,
    "url": "http://www.pandora.com/van-halen/listen",
  }
}
</script>
```

Action

Thing > Action

An action that can be performed (or has been performed, depending on its “actionStatus”) by a direct agent and indirect participants upon an object.

An action happens at a location with the help of an inanimate instrument.

The execution of the action is handled by an “actionHandler“ which “expects” parameters and may produce a “result”.

Specific action sub-type documentation specifies the exact expectation of each argument/role.

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 their 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*.
expects	SupportedClass	A type of the instance expected/taken by this action.

returns	SupportedClass	The type of the instance returned by this action.

Example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Event"
  "@id": "http://www.eventbrite.com/events/123",
  "name": "Come to my party!"
  "operation": {
    "@type": "RsvpAction"
    “actionStatus”: “PROPOSED”,
    "actionHandler" : {
      "@type": "WebPageHandler",
    }
  }
}
</script>
```

ActionStatus

Thing > Intangible > Enumeration > ActionStatus

The status of an Action.

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

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

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "FoodEstablishmentOrder"
  "url": "http://www.grubhub.com/restaurants/cascal/orders/1123",
  "operation": {
    "@type": "ConfirmAction"
    “actionStatus”: “PROPOSED”,
  }
}
</script>
```

ActionHandler

Thing > Intangible > ActionHandler

A mechanism to invoke and fulfill an action.

Example

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "ParcelDelivery",
  "deliveryAddress": {
    "@type": "PostalAddress",
    "streetAddress": "24 Willie Mays Plaza",
    "addressLocality": "San Francisco",
    "addressRegion": "CA",
    "addressCountry": "US",
    "postalCode": "94107"
  },
  "expectedArrival": "2013-03-12T12:00:00-08:00",
  "carrier": {
    "@type": "Organization",
    "name": "FedEx"
  },
  "itemShipped": {
    "@type": "Product",
    "name": "iPod Mini"
  },
  "partOfOrder": {
    "@type": "Order",
    "orderNumber": "176057",
    "seller": {
      "@type": "Person",
      "name": "Bob Dole"
    }
  },
  "operation": {
    "@type": "TrackAction",
    "actionHandler": {
      "@type": "WebPageHandler",
      "url": "http://fedex.com/track/1234567890"
    }
  }
}
</script>
```

WebPageHandler

Thing > Intangible > ActionHandler > WebPageHandler

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

Property	Type	Description

Example (reference)

```
<script type="application/ld+json">
```

```
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Check-In to your flight reservation",
  "about": {
    "@type": "Flight",
    "@id": "https://aa.com/flights/AA123"
    "name": "AA123",
    "operation": {
      "@type": "CheckInAction",
      "actionHandler": {
        "@type": "WebPageHandler",
        "url": "https://aa.com/flights/AA123/checkin"
      }
    }
  }
}
</script>
```

ApplicationHandler

Thing > Intangible > ActionHandler > ApplicationHandler

The Action Handler that allows performing actions through native applications.

Property	Type	Description
application	SoftwareApplication	The target application of the action handler.
minVersion	Text	The minimum version of the target application required to support this action.

WindowsActionHandler

Thing > Intangible > ActionHandler > ApplicationHandler > WindowsActionHandler

The Action Handler that allows performing actions through Windows applications.

Example:

```
<span itemscope itemtype="http://schema.org/Restaurant"
itemid="http://www.urbanspoon.com/r/1/5609/restaurant/Ballard/Rays-Boathouse-Seattle">
  <span itemprop="operation" itemscope itemtype="http://schema.org/ViewAction">
```

```
    <span itemprop="actionHandler" itemscope
itemtype="http://schema.org/WindowsActionHandler">
      <meta itemprop="application" content="msApplication://78901">
      <meta itemprop="minVersion" content="2.2.0.12">
    </span>
  </span>
</span>
```

WindowsPhoneActionHandler

Thing > Intangible > ActionHandler > ApplicationHandler > WindowsPhoneActionHandler

The Action Handler that allows performing actions through Windows Phone applications.

```
<span itemscope itemtype="http://schema.org/Restaurant"
itemid="http://www.urbanspoon.com/r/1/5609/restaurant/Ballard/Rays-Boathouse-Seattle">
  <span itemprop="operation" itemscope itemtype="http://schema.org/ViewAction">
    <span itemprop="actionHandler" itemscope
itemtype="http://schema.org/WindowsPhoneActionHandler">
      <meta itemprop="application" content="msApplication://abcd">
    </span>
  </span>
</span>
```

AndroidHandler

Thing > Intangible > ActionHandler > ApplicationHandler > AndroidHandler

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

An intent is dispatched combining the resource data URI as well as the action namespace.

For example:

Intent intent = new Intent();
intent.setAction(Intent.LISTEN);
intent.setData(“http://example.com/songs/123”);
startActivity(intent);

Example:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Restaurant",
  "@id": "http://opentable.com/cascal",
  "orders": {
    "@type": "ItemList",
    "@id": "http://opentable.com/cascal/orders",
    "operation": {
      "@type": "CreateAction",
      "actionHandler": {
        "@type": "AndroidHandler",
        "application":
"https://play.google.com/store/apps/details?id=com.opentable"
      }
    }
  }
}
</script>
```

Advanced concepts

HttpHandler

Thing > Intangible > ActionHandler > HttpHandler

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

If there is a “returns” class in the handler, there is an expectation that the response in the HTTP request will come back with an instance of that class.

Here is an example of what a serialization of the request looks like:

HTTP request headers of a HttpHandler invocation

```
Request URL:http://code.sgo.to/products/123
Request Method:POST
Status Code:200 OK
Request Headers
Accept:*/*
Accept-Encoding:gzip,deflate,sdch
Accept-Language:en-US,en;q=0.8,pt;q=0.6
Connection:keep-alive
```

```
Content-Length:75
Content-type:application/x-www-form-urlencoded
Host:code.sgo.to
Origin:http://code.sgo.to
Form Data
@type:ReviewAction
reviewBody:Awesome product!
reviewRating.ratingValue:5
```

Property	Type	Description
httpMethod	Text	POST or GET.

Example (reference):

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Approval request",
  "about": {
    "@type": "ExpenseReport",
    "description": "John's $10.13 expense for office supplies",
    "operation": {
      "@type": "ApproveAction",
      "handler": {
        "@type": "HttpActionHandler",
        "url": "https://myexpenses.com/approve?expenseId=abc123"
      }
    }
  }
}
</script>
```

SupportedClass

Thing > Intangible > SupportedClass

A class known to be support some properties. SupportedClasses are used to describe types that are expected in API calls while handling Actions (e.g. an API call may “require” that a numerical “rating” is passed while handling a ReviewAction).

Property	Type	Description
subClassOf	Class	The class which property’s support is being specified.
supportedProperty	Property, SupportedProperty	The properties that are supported in this class. A Property alone defaults to being a “required” SupportedProperty.

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "Please rate your experience.",
  "about": {
    "@type": “FoodEstablishment”,
    "@id": "http://reviews.com/restaurants/123",
    "name": "Joe's Diner"
    "operation": {
      "@type": "ReviewAction",
      "expects": {
        "@type": "SupportedClass",
        "supportedProperty": [
          {
            "@type": "SupportedProperty",
            "property": "http://schema.org/reviewBody",
            "required": "true",
          }, {
           "@type": "SupportedProperty",
            "property": "http://schema.org/reviewRating",
            "rangeIncludes": {
              "@type": "SupportedClass",
              "supportedProperty": {
                "property": "http://schema.org/ratingValue",
                "minValue": "1",
                "maxValue": "5",
             },
           },
        }],
      },
      "actionHandler": {
        "@type": "HttpActionHandler",
        "method": "POST"
      }
    }
  }
}
</script>
```

SupportedProperty

Thing > Intangible > SupportedProperty

A property known to be supported by a class. SupportedProperties are used to describe properties that are expected in API calls while handling Actions (e.g. an API call may “require” that a numerical “rating” is passed while handling a ReviewAction).

Property	Type	Description
property	Property	The property that the support is being specified.
required	boolean	Whether the property is required or not.
minValue	Number	The minimum value the property should have.
maxValue	Number	The maximum value the property should have.

Example

```
{
  "@context": "http://schema.org",
  "@type": "Airline",
  "@id": "http://code.sgo.to/airlines/123",
  "name": "American Airlines",
  "flights": {
    "@type": "ItemList",
    "@id": "http://code.sgo.to/airlines/123/flights",
    "name": "The AA flights",
    "operation": [{
        "@type": "SearchAction",
        "name": "Search available flights",
        "actionStatus": "proposed",
        "expects": {
          "@type": “SupportedClass”,
          "supportedProperty": [{
            "@type": "SupportedProperty",
            "property": "http://schema.org/query",
            "required": "true",
          }],
         },
         "actionHandler": {
           "@type": "HttpHandler",
           "name": "Searches for current flights",
           "httpMethod": "post"
         }
       }
     ]
   }
}
```

WebParamsHandler

Thing > Intangible > ActionHandler > WebPageHandler > WebParamsHandler

Property	Type	Description
httpMethod	Text	GET or POST.

WebFormHandler

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>
```

Action Hierarchy

- 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