



# D3.4v0.2. WSMO Use Case: Amazon E-commerce Service

WSMO Working Draft 13 January 2006

**This version:**

<http://www.wsmo.org/TR/d3/d3.4/v0.2/20060113/>

**Latest version:**

<http://www.wsmo.org/TR/d3/d3.4/v0.2/>

**Previous version:**

<http://www.wsmo.org/TR/d3/d3.4/v0.2/20051205/>

**Editors:**

Jacek Kopecky  
Dumitru Roman  
James Scicluna

**Co-author:**

Dieter Fensel

This document is also available in non-normative [PDF](#) version.

Copyright © 2005 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

---

## Table of contents

- [1. Introduction](#)
- [2. Amazon E-commerce Service Functionality](#)
- [3. WSMO Ontology for the Amazon E-commerce Service Datatypes](#)
- [4. WSMO Capability of the Amazon E-commerce Service](#)
  - [4.1 WSMO Web Service Definition for the Amazon E-commerce Service](#)
  - [4.2 Pre-Conditions](#)
  - [4.3 Post-Conditions](#)
- [5. WSMO Choreography for the Amazon E-commerce Service](#)
  - [5.1 State Signature](#)
  - [5.2 Transition Rules](#)
- [6. Conclusions](#)
- [Appendix A. The Complete WSMO Description for the Amazon E-commerce Service](#)
- [Acknowledgements](#)

---

## 1. Introduction

Amazon Web Services provides developers with direct access to [Amazon](#)'s technology platform. Build on Amazon's suite of Web services to enable and enhance your applications (from [Amazon Web Services website](#)). Currently, [Amazon Web Services consist of 5 Web services](#), i.e. 5 WSDL files:

- [Alexa Web Information Service](#) offers developers a platform for creating applications based on Alexa's repository of information about the Web. Its WSDL description can be found [here](#).

- [Amazon E-Commerce Service](#) (ECS) exposes Amazon's product data and e-commerce functionality. This allows developers, web site owners and merchants to leverage the data and functionality that Amazon uses to power its own e-commerce business. Its WSDL description can be found [here](#).
- [Amazon Historical Pricing](#) Web service gives developers programmatic access to over three years of sales data for books, music, videos, and DVDs (as sold by third-party sellers on Amazon.com). Sellers can use Amazon Historical Pricing to make informed decisions on pricing and purchasing. The WSDL description can be found [here](#).
- [Amazon Mechanical Turk](#) provides a Web services API for computers to integrate Artificial Intelligence directly into their processing. The WSDL description can be found [here](#).
- [Amazon Simple Queue Service](#) (Beta) offers a reliable, highly scalable hosted queue for buffering messages between distributed application components. Using the Simple Queue Service (SQS), developers can decouple components of their application so that they run independently. SQS provides the message management between the independent components. Any component of a distributed application can store any type of data in a reliable queue at Amazon.com. Another component or application can retrieve the data using queue semantics. Its WSDL description can be found [here](#).

In this document, we focus only on the Amazon E-Commerce Service (ECS); it is the most complex amongst the 5 services presented above, and is the only one which exposes e-commerce functionality. More specifically, Amazon E-Commerce Service (ECS) exposes Amazon's product data and e-commerce functionality. This allows developers, web site owners and merchants to leverage the data and functionality that Amazon uses to power its own e-commerce business (from [Amazon E-commerce Service website](#)).

In this document, we take a bottom-up approach to describing the Amazon E-commerce Service in WSMO. We start from the WSDL description of the service's interface, accompanied by the on-line documentation, and we distill the relevant WSMO descriptions from it. While the ECS contains five distinct groups of operations (as detailed below in [Section 2](#)) which could form five separate WSDL Web services, Amazon has chosen to publish them together as one service. In WSMO we follow this direction, even though the WSMO description could be split in the five sub-services without regard to how the WSDL description is structured.

In the rest of the document, we first describe the Amazon E-commerce Service in [Section 2](#), then we show how the WSMO ontologies are extracted from the XML Schema types of the WSDL description (in [Section 3](#)) then we talk about modeling the capability of the service in WSMO in [Section 4](#) and we go on to describe the WSMO Web Service Interface (especially the choreography) in [Section 5](#), finally presenting our conclusions in [Section 6](#). The [Appendix A](#) we present the complete WSMO description for the Amazon E-Commerce Service (ECS) that resulted from the analysis presented in this document.

## 2. Amazon E-commerce Service Functionality

The list of functionalities of the ECS (from [here](#) and [here](#)):

- Searching the Amazon catalog
- Looking up data for specific products
- Getting seller feedback for non-Amazon vendors
- Setting up shopping carts
- Looking up customers' wish lists and registries

Through ECS, you can access:

- Product data, including information about product availability and pricing for items in the Amazon catalog.
- Content from customers, like reviews, wish lists, and listmania lists.
- Seller information, like general information and customer feedback about the wide range of vendors on the Amazon site.

In ECS, you can also manage shopping carts of products for purchase through Amazon. This allows you to receive commissions on sales that originate with your Web site or application. You

have to transfer the customer to Amazon to complete the purchase, though; purchasing directly through ECS is **not supported**. ECS is a read-only system — that is, product data cannot be sent back to Amazon via ECS. The only data that is transmitted back to Amazon is information about the shopping cart and its contents, to allow the customer to complete their purchase. So you cannot, for example, use ECS to allow customers to create wish lists or reviews and submit them to the Amazon Web site.

The operations of ECS can be categorized as operations for Product data inquiry, Shopping cart manipulation, Customer content inquiry, Seller information inquiry and Other operations. These categories are shortly described in the following subsections ([source](#)).

## 2.1 Product Data

Product data operations allow you to look up Amazon products, either by searching for groups of products or by looking up specific items.

ECS Operation	Description
<a href="#">ItemSearch</a>	Search the product database using a one of a variety of possible criteria.
<a href="#">ItemLookup</a>	Look up specific product(s) in the Amazon database.
<a href="#">SimilarityLookup</a>	Look up products that are similar to specific other products.
<a href="#">SellerListingSearch</a>	Search for seller product listings (zShops and Marketplace).
<a href="#">SellerListingLookup</a>	Look up a specific zShops or Marketplace product listing from a particular seller.

## 2.2 Shopping Cart

[The shopping cart operations](#) allow you to create a shopping cart of items and hand it off to Amazon so the customer can purchase the selected items.

ECS Operation	Description
<a href="#">CartCreate</a>	Create a shopping cart and add an item(s).
<a href="#">CartAdd</a>	Add products to the shopping cart
<a href="#">CartModify</a>	Change quantities or status of entries in the shopping cart.
<a href="#">CartClear</a>	Clear all entries in a shopping cart
<a href="#">CartGet</a>	Get the contents of a shopping cart with updated price and availability information.

## 2.3 Customer Content

[Customer content operations](#) allow you to look up publicly available customer content, such as reviews, wish lists and Listmania lists.

ECS Operation	Description
<a href="#">CustomerContentSearch</a>	Search for customers.
<a href="#">CustomerContentLookup</a>	Look up specific customer(s) information (location/name/reviews).
<a href="#">ListLookup</a>	Look up specific list(s) created by customers.
<a href="#">ListSearch</a>	Search for customer-created lists.

## 2.4 Seller Information

ECS allows you to look up feedback on specific sellers and get seller product listings.

ECS Operation	Description
<a href="#">SellerLookup</a>	Look up information about a specific seller.

## 2.5 Other Operations

ECS Operation	Description
<a href="#">BrowseNodeLookup</a>	Returns information about a browse node, including child nodes.
<a href="#">Help</a>	Returns information about how to use an ECS operation or about what to expect from ECS response groups.
<a href="#">TransactionLookup</a>	Look up the amount of a specific transaction.

## 3. WSMO Ontology for the Amazon E-commerce Service Datatypes

This section explains how the ontologies are extracted from the XML Schema types of the WSDL description of Amazon. We present here a one-to-one mapping between the Schema types and the concepts without considering further modelling which can be done using ontologies. In particular, the data structures only contain binary relationships which we model as concept attributes, therefore the presented ontology contains no relations.

The XML types considered below are the ones used by the messages within the operations. All the operations and data types are described in the [Amazon E-commerce Service WSDL document](#).

### Help Request

Instances of this concept are consumed as input for the *help* operation. It allows to define the keywords related to the help and the type of help needed as shown in Listing 1.

Listing 1. HelpRequest Type

```
<xs:complexType name="HelpRequest">
  <xs:sequence>
    <xs:element name="About" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="HelpType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Operation"/>
          <xs:enumeration value="ResponseGroup"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The equivalent "helpRequest" concept is shown in Listing 2 below. An axiom to limit the type of help is also defined.

```

concept helpRequest
  nonFunctionalProperties
    dc#description hasValue "A help request"
  endNonFunctionalProperties
  about ofType _string
  helpType ofType (1) _string
  responseGroup ofType _string

axiom helpTypeConstraint
  nonFunctionalProperties
    dc#description hasValue "The type of help can only be 'Operation' or 'ResponseGroup'"
  endNonFunctionalProperties
  definedBy !-
    ?x memberof helpRequest and
    ?x[helpType hasValue ?type] and
    (
      ?type hasValue "Operation" or
      ?type hasValue "ResponseGroup"
    ).

```

## Help Response

Instance of this type are returned by the *Help* operation. It encloses information about the help requested as shown in Listing 3.

```

<xs:element name="HelpResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Information" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Information">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element ref="tns:OperationInformation" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="tns:ResponseGroupInformation" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="OperationInformation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="RequiredParameters" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Parameter" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="AvailableParameters" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Parameter" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="DefaultResponseGroups" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ResponseGroup" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="AvailableResponseGroups" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ResponseGroup" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ResponseGroupInformation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="CreationDate" type="xs:string" minOccurs="0"/>
      <xs:element name="ValidOperations" minOccurs="0">
        <xs:complexType>
          <xs:sequence>

```

The "helpResponse" concept defines the response for a help request. The modelling to represent the two types of information takes a different shape than that of the schema types defined above. Rather than having an "information" concept which defines both "responseGroupInformation" and "operationInformation" as attributes, the latter two are modelled as sub-classes of the former. These concepts are defined in Listing 4 below.

Listing 4. helpResponse, information, operationInformation and responseGroupInformation Concepts

```

concept helpResponse
  nonFunctionalProperties
    dc#description hasValue "A help response object"
  endNonFunctionalProperties
  informationList impliesType information
  responseType ofType (1) _string

concept information
  nonFunctionalProperties
    dc#description hasValue "Defines the superclass for operationInformation
      and responseGroupInformation"
  endNonFunctionalProperties
  name ofType (1) _string

concept operationInformation subConceptOf information
  nonFunctionalProperties
    dc#description hasValue "Help about an operation"
  endNonFunctionalProperties
  description ofType _string
  requiredParameters ofType _string
  availableParameters ofType _string
  defaultResponseGroups ofType _string
  availableResponseGroups ofType _string

concept responseGroupInformation subConceptOf information
  nonFunctionalProperties
    dc#description hasValue "Help about a Response Group"
  endNonFunctionalProperties
  creationDate ofType (1) _string
  validOperations ofType _string
  elements ofType _string

```

## Item Search Request

This type is used to define search requests by keywords and is used as an input for the operation *ItemSearch*. Amazon allows to search items that do not necessarily relate to books. Thus, such keywords for searching include also actor names, categories and brands.

```

<xs:complexType name="ItemSearchRequest">
  <xs:sequence>
    <xs:element name="Actor" type="xs:string" minOccurs="0"/>
    <xs:element name="Artist" type="xs:string" minOccurs="0"/>
    <xs:element ref="tns:AudienceRating" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="Author" type="xs:string" minOccurs="0"/>
    <xs:element name="Brand" type="xs:string" minOccurs="0"/>
    <xs:element name="BrowseNode" type="xs:string" minOccurs="0"/>
    <xs:element name="City" type="xs:string" minOccurs="0"/>
    <xs:element name="Composer" type="xs:string" minOccurs="0"/>
    <xs:element ref="tns:Condition" minOccurs="0"/>
    <xs:element name="Conductor" type="xs:string" minOccurs="0"/>
    <xs:element name="Count" type="xs:positiveInteger" minOccurs="0"/>
    <xs:annotation>
      <xs:appinfo>
        <aws-se:restricted
          xmlns:aws-se="http://webservices.amazon.com/AWS-SchemaExtensions">
            <aws-se:excludeFrom>public</aws-se:excludeFrom>
            <aws-se:excludeFrom>partner</aws-se:excludeFrom>
          </aws-se:restricted>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="Cuisine" type="xs:string" minOccurs="0"/>
    <xs:element ref="tns:DeliveryMethod" minOccurs="0"/>
    <xs:element name="Director" type="xs:string" minOccurs="0"/>
    <xs:element name="FutureLaunchDate" type="xs:string" minOccurs="0"/>
    <xs:element name="ISPUPostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="ItemPage" type="xs:positiveInteger" minOccurs="0"/>
    <xs:element name="Keywords" type="xs:string" minOccurs="0"/>
    <xs:element name="Manufacturer" type="xs:string" minOccurs="0"/>
    <xs:element name="MaximumPrice" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="MerchantId" type="xs:string" minOccurs="0"/>
    <xs:element name="MinimumPrice" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="MusicLabel" type="xs:string" minOccurs="0"/>
    <xs:element name="Neighborhood" type="xs:string" minOccurs="0"/>
    <xs:element name="Orchestra" type="xs:string" minOccurs="0"/>
    <xs:element name="PostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="Power" type="xs:string" minOccurs="0"/>
    <xs:element name="Publisher" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="SearchIndex" type="xs:string" minOccurs="0"/>
    <xs:element name="Sort" type="xs:string" minOccurs="0"/>
    <xs:element name="State" type="xs:string" minOccurs="0"/>
    <xs:element name="TextStream" type="xs:string" minOccurs="0"/>
    <xs:element name="Title" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

Not all the elements in the type are translated to ontologies. Some of these elements are related to the internal workings of the Web service and thus one doesn't need to specify such information at the ontological level.

```

concept itemSearchRequest
  nonFunctionalProperties
    dc:description hasValue "A request to search for an item"
  endNonFunctionalProperties
  actor ofType _string
  artist ofType _string
  rating ofType ratings#audienceRating
  author ofType _string
  brand ofType (1) _string
  city ofType _string
  composer ofType _string
  conductor ofType _string
  count ofType _int
  cuisine ofType _string
  delivery ofType (1) tasks#deliveryMethod
  director ofType _string
  futureLaunchDate ofType _string
  ispuPostalCode ofType (1) _string
  keywords ofType _string
  manufacturer ofType _string
  maxPrice ofType (1) _decimal
  merchantId ofType (1) _string
  minPrice ofType (1) _decimal
  musicLabel ofType _string
  neighbourhood ofType _string
  orchestra ofType _string
  postalCode ofType (1) _string
  power ofType _string
  publisher ofType _string
  responseGroup ofType _string
  sort ofType (1) _string
  state ofType _string
  title ofType (1) _string

```

## Item Search Response

This particular data type is returned by the *ItemSearch* operation. It encloses a set of items which match the constraints specified by the user in the search request.

Listing 7. ItemSearchResponse Type

```

<xs:element name="ItemSearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Items" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Again, the element *OperationRequest* is related to the internal working of the web service and thus it's not specified by the ontology. Also, rather than specifying the concept "itemSearchResponse", the concept "itemContainer" is directly used to specify the items returned by such a method. This will also allow to reuse the concept within other contexts (such as the result of an "ItemLookupRequest").

```

concept itemContainer
  nonFunctionalProperties
    dc:description hasValue "Contains a list of items"
  endNonFunctionalProperties
  items ofType item

```

Following item container, the *Item* datatype is defined below.

Listing 9. ItemSearchResponse Type

```

<xs:element name="Item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ASIN" type="xs:string"/>
      <xs:element ref="tns:Errors" minOccurs="0"/>
      <xs:element name="DetailPageURL" type="xs:string" minOccurs="0"/>
      <xs:element name="SalesRank" type="xs:string" minOccurs="0"/>
      <xs:element name="SmallImage" type="tns:Image" minOccurs="0"/>
      <xs:element name="MediumImage" type="tns:Image" minOccurs="0"/>
      <xs:element name="LargeImage" type="tns:Image" minOccurs="0"/>
      <xs:element ref="tns:ItemAttributes" minOccurs="0"/>
      <xs:element ref="tns:OfferSummary" minOccurs="0"/>
      <xs:element ref="tns:Offers" minOccurs="0"/>
      <xs:element ref="tns:VariationSummary" minOccurs="0"/>
      <xs:element ref="tns:Variations" minOccurs="0"/>
      <xs:element ref="tns:CustomerReviews" minOccurs="0"/>
      <xs:element ref="tns:EditorialReviews" minOccurs="0"/>
      <xs:element ref="tns:SimilarProducts" minOccurs="0"/>
      <xs:element ref="tns:Accessories" minOccurs="0"/>
      <xs:element ref="tns:Tracks" minOccurs="0"/>
      <xs:element ref="tns:BrowseNodes" minOccurs="0"/>
      <xs:element ref="tns:SearchInside" minOccurs="0"/>
      <xs:element ref="tns:PromotionalTag" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The "item" concept in Listing 10 does not define all of the elements described above (for example "SearchInside") since they are related to the internal working of the web service. Furthermore, we envision that certain types of concepts (such as customer reviews and tracks) would be defined in external domain ontologies. Note also that we define the notion of "minimalltem" which defines an item by its id and title. This concept is used to identify "extra" attributes associated with a generic item. These attributes are similar products and accessories.

```

concept item
  nonFunctionalProperties
    dc:description hasValue "An item handled by the Amazon Service"
  endNonFunctionalProperties
  asin ofType _string
  irl ofType _iri
  salesRank ofType _string
  smallImage ofType graphics#image
  mediumImage ofType graphics#image
  largeImage ofType graphics#image
  offerSummary ofType commerce#offerSummary
  offers ofType commerce#offer
  variationSummary ofType commerce#variationSummary
  variations ofType commerce#variation
  customerReviews ofType ratings#customerReview
  editorialReviews ofType ratings#editorialReview
  similarProduct ofType minimalItem
  accessories ofType minimalItem
  tracks ofType music#track
    node ofType browseNode
  promotionalTag ofType _string

concept minimalItem
  nonFunctionalProperties
    dc:description hasValue "Defines the minimal information needed
      for defining an item"
  endNonFunctionalProperties
  asin ofType _string
  title ofType _string

```

### Item Lookup Request

This datatype is used by the *ItemLookup* operation which allows to search for an item by its identifier.

```

<xs:complexType name="ItemLookupRequest">
  <xs:sequence>
    <xs:element ref="tns:Condition" minOccurs="0"/>
    <xs:element ref="tns:DeliveryMethod" minOccurs="0"/>
    <xs:element name="FutureLaunchDate" type="xs:string" minOccurs="0"/>
    <xs:element name="IdType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ASIN"/>
          <xs:enumeration value="UPC"/>
          <xs:enumeration value="SKU"/>
          <xs:enumeration value="EAN"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ISPUPostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="MerchantId" type="xs:string" minOccurs="0"/>
    <xs:element name="OfferPage" type="xs:positiveInteger" minOccurs="0"/>
    <xs:element name="ItemId" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ReviewPage" type="xs:positiveInteger" minOccurs="0"/>
    <xs:element name="SearchIndex" type="xs:string" minOccurs="0"/>
    <xs:element name="SearchInsideKeywords" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:appinfo>
          <aws-se:restricted
            xmlns:aws-se="http://webservices.amazon.com/AWS-SchemaExtensions">
            <aws-se:excludeFrom>public</aws-se:excludeFrom>
            <aws-se:excludeFrom>partner</aws-se:excludeFrom>
          </aws-se:restricted>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="VariationPage" type="xs:positiveInteger" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

Similarly as for previous data types, the concept describing the *ItemLookupRequest* type does not define all of the above attributes. However, the ontology defines an axiom that restricts the values of the type of ID requested by the user as shown below in Listing 12.

Listing 12. itemLookupRequest Concept and typeOfId axiom definition

```

concept itemLookupRequest
  nonFunctionalProperties
    dc#description hasValue "A request to lookup for an item using an ASIN"
    dc#relation hasValue typeOfId
  endNonFunctionalProperties
  delivery ofType (1) tasks#deliveryMethod
  futureLaunchDate ofType _string
  idType ofType (1) _string
  ispuPostalCode ofType (0 1) _string
  merchantId ofType (0 1) _string
  id ofType (1) _string
  responseGroup ofType _string

axiom typeOfId
  nonFunctionalProperties
    dc#description hasValue "There are only four types of IDs: ASIN,

```

```

UPC, SKU, EAN"
endNonFunctionalProperties
definedBy !-
  ?x memberof itemLookupRequest and
  ?x[idType hasValue ?type] and
  (
    ?type hasValue "ASIN" or
    ?type hasValue "UPC" or
    ?type hasValue "SKU" or
    ?type hasValue "EAN"
  )

```

## Item Lookup Response

This message is returned by the *ItemLookup* operation. It has the same structure as for *ItemSearchResponse* in that it returns a list of items. Hence, the "itemContainer" concept defined in Listing 6 is reused for describing this message type. The datatype is defined in Listing 13 below:

Listing 13. ItemLookupResponse Type

```

<xs:element name="ItemLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Items" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Browse Node Lookup Request

This message is used as an input by the *BrowseNode* operation. It defines a node identifier and a response group as follows:

Listing 14. BrowseNodeLookupRequest Type

```

<xs:complexType name="BrowseNodeLookupRequest">
  <xs:sequence>
    <xs:element name="BrowseNodeid" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent concept is defined in Listing 15.

```

concept browseNodeLookupRequest
  nonFunctionalProperties
    dc:description hasValue "Allows to browse through a node"
  endNonFunctionalProperties
  nodeId ofType (1) _string
  responseGroup ofType _string

```

## Browse Node Lookup Response

This is the output of the *BrowseNode* operation. It defines a list of nodes and references to their children which can be browsed by the client. The related datatypes are defined in Listing 16.

Listing 16. BrowseNodeLookupResponse, BrowseNodes and BrowseNode Types

```

<xs:element name="BrowseNodeLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:BrowseNodes" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="BrowseNodes">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element ref="tns:BrowseNode" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="BrowseNode">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BrowseNodeId" type="xs:string" minOccurs="0"/>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="Children" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="tns:BrowseNode" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Ancestors" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="tns:BrowseNode" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Rather than specifying an intermediary "browseNodes" concept, the concept "browseNodeLookupResponse" defines an attribute in which any number of "browseNode" elements can be defined. The related concepts are defined in Listing 17.

```

concept browseNodeLookupResponse
nonFunctionalProperties
  dc:description hasValue "List of nodes returned by a request
    to lookup a node to browse"
endNonFunctionalProperties
  nodes ofType browseNode

concept browseNode
nonFunctionalProperties
  dc:description hasValue "Node information"
endNonFunctionalProperties
  nodeId ofType (1) _string
  name ofType _string
  children ofType browseNode
  ancestors ofType browseNode

```

## List Search Request

This datatype specifies constraints to allow searching for lists (wish-list, wedding registry or baby registry). Instances of this datatype are accepted as input by the *ListSearch* operation and is specified in Listing 18.

Listing 18. ListSearchRequest Type

```

<xs:complexType name="ListSearchRequest">
  <xs:sequence>
    <xs:element name="City" type="xs:string" minOccurs="0"/>
    <xs:element name="Email" type="xs:string" minOccurs="0"/>
    <xs:element name="FirstName" type="xs:string" minOccurs="0"/>
    <xs:element name="LastName" type="xs:string" minOccurs="0"/>
    <xs:element name="ListPage" type="xs:positiveInteger" minOccurs="0"/>
    <xs:element name="ListType">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="WishList"/>
          <xs:enumeration value="WeddingRegistry"/>
          <xs:enumeration value="BabyRegistry"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="State" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent concept "listSearchRequest" is shown in Listing 19 below. A restriction on the type of the list in the form of an axiom is also specified. Note that the concept "typeOfList" is reused within other contexts and the constraint is associated to this concept.

```
concept listSearchRequest
nonFunctionalProperties
  dc:description hasValue "Allows to search for a list"
endNonFunctionalProperties
city ofType (0 1) _string
email ofType _string
firstName ofType (0 1) _string
lastName ofType (0 1) _string
listType ofType (0 1) typeOfList
name ofType (0 1) _string
responseGroup ofType _string
state ofType _string

concept typeOfList
nonFunctionalProperties
  dc:description hasValue "Defines the type of lists and relates
    to a restriction on the value"
  dc:relation hasValue listTypeConstraint
endNonFunctionalProperties
value ofType (1) _string

axiom listTypeConstraint
nonFunctionalProperties
  dc:description hasValue "A list type can be 'WishList',
    'WeddingRegistry' and 'BabyRegistry'"
endNonFunctionalProperties
definedBy !-
  ?x memberof typeOfList and
  ?x[value hasValue ?type] and
  (
    ?type hasValue "WishList" or
    ?type hasValue "WeddingRegistry" or
    ?type hasValue "BabyRegistry"
  ).
```

## List Search Response

Instances of this datatype are the output result of the *ListSearch* operation.

```

<xs:element name="ListSearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Lists" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Lists">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element name="TotalResults" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="TotalPages" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element ref="tns:List" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="List">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ListId" type="xs:string"/>
      <xs:element name="ListURL" type="xs:string" minOccurs="0"/>
      <xs:element name="RegistryNumber" type="xs:string" minOccurs="0"/>
      <xs:element name="ListName" type="xs:string"/>
      <xs:element name="ListType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="WishList"/>
            <xs:enumeration value="WeddingRegistry"/>
            <xs:enumeration value="BabyRegistry"/>
            <xs:enumeration value="Listmania"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="TotalItems" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="TotalPages" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="DateCreated" type="xs:string" minOccurs="0"/>
      <xs:element name="OccasionDate" type="xs:string" minOccurs="0"/>
      <xs:element name="CustomerName" type="xs:string" minOccurs="0"/>
      <xs:element name="PartnerName" type="xs:string" minOccurs="0"/>
      <xs:element name="AdditionalName" type="xs:string" minOccurs="0"/>
      <xs:element name="Comment" type="xs:string" minOccurs="0"/>
      <xs:element name="Image" type="tns:Image" minOccurs="0"/>
      <xs:element name="AverageRating" type="xs:decimal" minOccurs="0"/>
      <xs:element name="TotalVotes" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="TotalTimesRead" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element ref="tns:ListItem" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ListItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ListItemId" type="xs:string" minOccurs="0"/>
      <xs:element name="DateAdded" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The extracted concepts are defined as "listContainer" (describing a container for lists), "list" (describing the information related to a list) and "listItem" (an item of a list). The "list" concept is related also to the axiom "listTypeConstraint" which restricts the type of lists.

Listing 21. listContainer, list and listItem Concepts

```

concept listContainer
  nonFunctionalProperties
    dc#description hasValue "Defines a container for lists"
  endNonFunctionalProperties
  lists ofType list

concept list
  nonFunctionalProperties
    dc#description hasValue "Defines information related to a list"
    dc#relation hasValue listTypeConstraint
  endNonFunctionalProperties
  listId ofType (1) _string
  listUrl ofType (0 1) _iri
  registryNumber ofType (1) _string
  listName ofType (1) _string
  totalItems ofType (0 1) _int
  totalPages ofType (0 1) _int
  dateCreated ofType (0 1) _string
  customerName ofType (0 1) _string
  partnerName ofType (0 1) _string
  additionalName ofType _string
  comment ofType _string
  image ofType graphics#image
  averageRating ofType _decimal
  totalVotes ofType _int
  totalTimesRead ofType _int
  itemList ofType listItem
  listType ofType (1) typeOfList

concept listItem
  nonFunctionalProperties
    dc#description hasValue "Defines an item in a list"
    dc#relation hasValue listTypeConstraint
  endNonFunctionalProperties
  itemId ofType (1) _string
  dateAdded ofType (0 1) _string
  comment ofType _string
  quantityDesired ofType (0 1) _int
  quantityReceived ofType (0 1) _int
  items ofType item

```

## List Lookup Request

The operation *ListLookup* accepts instances of this datatype as an input. These instances define the information required to search for a List by its identifier as shown in Listing 22.

```

<xs:complexType name="ListLookupRequest">
  <xs:sequence>
    <xs:element ref="tns:Condition" minOccurs="0"/>
    <xs:element ref="tns:DeliveryMethod" minOccurs="0"/>
    <xs:element name="ISPUPostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="ListId" type="xs:string" minOccurs="0"/>
    <xs:element name="ListType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="WishList"/>
          <xs:enumeration value="Listmania"/>
          <xs:enumeration value="WeddingRegistry"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="MerchantId" type="xs:string" minOccurs="0"/>
    <xs:element name="ProductGroup" type="xs:string" minOccurs="0"/>
    <xs:element name="ProductPage" type="xs:positiveInteger"
      minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="Sort" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent "listLookupRequest" concept is defined in Listing 23. Similarly for other data types, not all the elements are mapped to attributes in the target concept.

```

concept listLookupRequest
nonFunctionalProperties
  dc#description hasValue "Allows to search for a list"
endNonFunctionalProperties
  delivery ofType (1) tasks#deliveryMethod
  ispuPostalCode ofType (1) _string
  listId ofType (1) _string
  listType ofType (1) typeOfList
  merchantId ofType (1) _string
  productGroup ofType _string
  productPage ofType (0 1) _int
  responseGroup ofType _string
  sort ofType _string

```

## List Lookup Response

Instances of this data type are produced as a result of the *ListLookup* operation. It defines the same structure as for the *ListSearchResponse* datatype and thus it is mapped to the "listContainer" concept.

Listing 24. ListLookupResponse Type

```

<xs:element name="ListLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Lists" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Customer Content Search Request

This datatype is taken as an input by the *CustomerContentSearch* operation . It defines information to allow to search for customers of Amazon.

Listing 25. CustomerContentSearchRequest Type

```

<xs:complexType name="CustomerContentSearchRequest">
  <xs:sequence>
    <xs:element name="CustomerPage" type="xs:positiveInteger"
      minOccurs="0"/>
    <xs:element name="Email" type="xs:string" minOccurs="0"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent concept "itemLookupRequest" is defined in Listing 26 below. Note that the customer page (of type integer) refers to an internal index of Amazon and is not thus modelled in the concept.

Listing 26. customerSearchRequest Concept

```

concept customerSearchRequest
  nonFunctionalProperties
    dc#description hasValue "Defines information to allow to search for an Amazon Customer"
  endNonFunctionalProperties
  email ofType _string
  name ofType (0 1) _string
  responseGroup ofType _string

```

## Customer Content Search Response

Defines a list of customers returned by the *CustomerContentSearch* operation.

Listing 27. CustomerContentSearchResponse Type

```

<xs:element name="CustomerContentSearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Customers" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The equivalent concept is defined below. Note that a customer information would be defined in an external domain ontology for commerce. Furthermore, a container concept is defined so that it can be reused within other contexts.

```

concept customerContainer
  nonFunctionalProperties
    dc#description hasValue "Defines a list of customers
      resulting from a search request"
  endNonFunctionalProperties
  customers ofType commerce#customer

```

## Customer Content Lookup Request

Allows to search for a customer using his or her identifier. This message is used as an input by the *CustomerContentLookup* operation.

Listing 29. CustomerContentLookupRequest Type

```

<xs:complexType name="CustomerContentLookupRequest">
  <xs:sequence>
    <xs:element name="CustomerId" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ReviewPage" type="xs:positiveInteger" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent concept is shown below in Listing 30.

Listing 30. customerLookupResponse Concept

```

concept customerLookupRequest
  nonFunctionalProperties
    dc#description hasValue "Defines information to allow to search for an
      Amazon Customer by an identifier"
  endNonFunctionalProperties
  customerId ofType (1) _string
  responseGroup ofType _string
  reviewPage ofType _int

```

## Customer Content Lookup Response

This datatype is the output of the *CustomerContentLookup* operation and is defined in Listing 31 below.

Listing 31. CustomerContentSearchRequest Type

```

<xs:element name="CustomerContentLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Customers" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The resulting translated concept is the same as for the *CustomerContentSearch* operation. Thus, the concept "customerContainer" is reused in this case. This concept is defined in Listing 28 above.

## Similarity Lookup Request

This datatype is used as an input by the *SimilarityLookup* operation. It allows to search for more

than one item (that is, more than one item identifier can be specified). If only one item identifier is given, the *SimilarityType* attribute is ignored. The attribute can take two values: *Intersection* and *Random*. The former forces the Web service to return items that are similar to *all* of the items in the request. The *Random* value forces the Web service to return an assortment of similar products corresponding to any of the items in the request. This datatype is depicted in Listing 32.

Listing 32. SimilarityLookupRequest Type

```
<xs:complexType name="SimilarityLookupRequest">
  <xs:sequence>
    <xs:element ref="tns:Condition" minOccurs="0"/>
    <xs:element ref="tns:DeliveryMethod" minOccurs="0"/>
    <xs:element name="ItemId" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ISPUPostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="MerchantId" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="SimilarityType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Intersection"/>
          <xs:enumeration value="Random"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

The translated concept requires to specify a restriction on the similarity type as shown in Listing 33 below.

```

concept similarityLookupRequest
  nonFunctionalProperties
    dc:description hasValue "Defines constraints to search for
      similar items for a particular product"
    dc:relation hasValue {similarityRestriction, similarItems}
  endNonFunctionalProperties
  delivery ofType tasks#deliveryMethod
  itemId ofType _string
  ispuPostalCode ofType (1) _string
  merchantId ofType (1) _string
  responseGroup ofType _string
  similarityType ofType _string

axiom similarityRestriction
  nonFunctionalProperties
    dc:description hasValue "The type of similarity can be
      'Intersection' or 'Random'"
  endNonFunctionalProperties
  definedBy !-
    ?x memberof similarityLookupRequest and
    ?x[similarityType hasValue ?type] and
    (
      ?type hasValue "Intersection" or
      ?type hasValue "Random"
    ).

relation similarItems(ofType item, ofType item)
  nonFunctionalProperties
    dc:description hasValue "Relation holds for items which
      are similar"
  endNonFunctionalProperties

```

## Similarity Lookup Response

This datatype is the output result of the *SimilarityLookup* operation. It returns a list of items as for the search by keywords and search by ID. Therefore, the concept "itemContainer" is reused in this case. The datatype is defined in Listing 34 below.

Listing 34. SimilarityLookupResponse Type

```

<xs:element name="SimilarityLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Items" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Seller Lookup Request

The operation *SellerLookup* takes instances of this type as input. It defines constraints to allow searching for a particular seller as shown in Listing 35. Note that the FeedbackPage element describes how many feedback entries (related to the seller) are returned.

Listing 35. SellerLookupRequest Type

```

<xs:complexType name="SellerLookupRequest">
  <xs:sequence>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"

```

```
        maxOccurs="unbounded"/>
<xs:element name="SellerId" type="xs:string" minOccurs="0"
  maxOccurs="unbounded"/>
<xs:element name="FeedbackPage" type="xs:positiveInteger"
  minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

The equivalent concept is defined in Listing 36.

Listing 36. sellerLookupRequest concept

```
concept sellerLookupRequest
nonFunctionalProperties
  dc:description hasValue "Defines constraints to search for a seller"
endNonFunctionalProperties
sellerId ofType (1) _string
responseGroup ofType _string
feedbackPage ofType _int
```

### **Seller Lookup Response**

This datatype is referred by the output of the *SellerLookup* operation. It specifies a list of Sellers that match the given criteria in the input. The relevant datatypes are shown in Listing 37.

```

<xs:element name="SellerLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Sellers" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Sellers">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element name="TotalResults" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="TotalPages" type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element ref="tns:Seller" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Seller">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SellerId" type="xs:string"/>
      <xs:element name="SellerName" type="xs:string" minOccurs="0"/>
      <xs:element name="Nickname" type="xs:string" minOccurs="0"/>
      <xs:element name="GlancePage" type="xs:string" minOccurs="0"/>
      <xs:element name="About" type="xs:string" minOccurs="0"/>
      <xs:element name="MoreAbout" type="xs:string" minOccurs="0"/>
      <xs:element name="Location" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="City" type="xs:string" minOccurs="0"/>
            <xs:element name="State" type="xs:string" minOccurs="0"/>
            <xs:element name="Country" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="AverageFeedbackRating"
        type="xs:decimal" minOccurs="0"/>
      <xs:element name="TotalFeedback"
        type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element name="TotalFeedbackPages"
        type="xs:nonNegativeInteger"
        minOccurs="0"/>
      <xs:element ref="tns:SellerFeedback" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Similarly for customers, the result is translated to one concept in such a way that the "sellers" attribute links to any number of "seller" concepts specified in a commerce ontology as shown in Listing 38.

```

concept sellerContainer
  nonFunctionalProperties
    dc:description hasValue "Defines a list of sellers"
  endNonFunctionalProperties
  sellers ofType commerce#seller

```

## Cart Get Request

Used as an input by the *CartGet* operation. It allows to specify an identifier of the requested cart. The *HMAC* element is an identifier for a security token and is also modelled in the ontology.

Listing 39. CartGetRequest Type

```

<xs:complexType name="CartGetRequest">
  <xs:sequence>
    <xs:element name="CartId" type="xs:string" minOccurs="0"/>
    <xs:element name="HMAC" type="xs:string" minOccurs="0"/>
    <xs:element name="MergeCart" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent concept is defined in Listing 40 below.

Listing 40. cartGetRequest concept

```

concept cartGetRequest
  nonFunctionalProperties
    dc:description hasValue "A request to get the Cart"
  endNonFunctionalProperties
  cartId ofType (1) _string
  hmac ofType (1) _string
  mergeCart ofType (1) _string
  responseGroup ofType _string

```

## Cart Get Response

Instances of this datatype are the resulting output of the *CartGet* operation. It serves as a container for the cart object being returned as shown in Listing 41. The *Cart* and *CartItem* datatypes are also shown for the sake of completeness.

Listing 41. CartGetResponse, Cart and CartItem Types

```

<xs:element name="CartGetResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Cart" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Cart">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element name="CartId" type="xs:string"/>
      <xs:element name="HMAC" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="URLEncodedHMAC" type="xs:string"/>
<xs:element name="PurchaseURL" type="xs:string"/>
<xs:element name="SubTotal" type="tns:Price" minOccurs="0"/>
<xs:element ref="tns:CartItems" minOccurs="0"/>
<xs:element ref="tns:SimilarProducts" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="CartItem">
<xs:sequence>
<xs:element name="CartItemid" type="xs:string"/>
<xs:element name="ASIN" type="xs:string" minOccurs="0"/>
<xs:element name="ExchangeId" type="xs:string" minOccurs="0"/>
<xs:element name="MerchantId" type="xs:string" minOccurs="0"/>
<xs:element name="SellerId" type="xs:string" minOccurs="0"/>
<xs:element name="SellerNickname" type="xs:string" minOccurs="0"/>
<xs:element name="Quantity" type="xs:string"/>
<xs:element name="Title" type="xs:string" minOccurs="0"/>
<xs:element name="ProductGroup" type="xs:string" minOccurs="0"/>
<xs:element name="ListOwner" type="xs:string" minOccurs="0"/>
<xs:element name="ListType" type="xs:string" minOccurs="0"/>
<xs:element name="Price" type="tns:Price" minOccurs="0"/>
<xs:element name="ItemTotal" type="tns:Price" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

```

Rather than creating a "cartGetResponse" concept, it is more convenient to use a "cart" concept as a mapping for this datatype. This will allow also reusability of this concept within other contexts.

```

concept cart
  nonFunctionalProperties
    dc#description hasValue "A cart is represented by an ID and a
      list of items"
  endNonFunctionalProperties
  id ofType (1) _string
  hmac ofType (1) _string
  urlEncodedHMAC ofType (1) _string
  purchaseUrl ofType (1) _iri
  subTotal ofType (1) commerce#price
  items ofType cartItem
  similarProducts ofType minimalItem

concept cartItem
  nonFunctionalProperties
    dc#description hasValue "A cart item is described by a
      specific item and a quantity"
  endNonFunctionalProperties
  id ofType (1) _string
  asin ofType (1) _string
  exchangeId ofType (1) _string
  merchantId ofType (1) _string
  sellerId ofType (1) _string
  sellerNickName ofType (1) _string
  quantity ofType (1) _int
  title ofType (1) _string
  productGroup ofType (1) _string
  listOwner ofType (1) _string
  listType ofType (1) _string
  item ofType (1) item
  unitPrice ofType (1) commerce#price
  totalPrice ofType (1) commerce#price

```

### Cart Add Request

Defines the required elements to add items to an existing cart. Instances of this datatype are the inputs of the *CartAdd* operation.

Listing 43. CartAddRequest Type

```

<xs:complexType name="CartAddRequest">
  <xs:sequence>
    <xs:element name="CartId" type="xs:string" minOccurs="0"/>
    <xs:element name="HMAC" type="xs:string" minOccurs="0"/>
    <xs:element name="MergeCart" type="xs:string" minOccurs="0"/>
    <xs:element name="Items" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ASIN" type="xs:string" minOccurs="0"/>
                <xs:element name="OfferListingId" type="xs:string"
                  minOccurs="0"/>
                <xs:element name="Quantity" type="xs:positiveInteger"
                  minOccurs="0"/>
                <xs:element name="AssociateTag" type="xs:string" minOccurs="0"/>
                <xs:element name="ListItemId" type="xs:string" minOccurs="0"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The respective concept is defined in Listing 44 below.

Listing 44. cartAddRequest concept

```

concept cartAddRequest
  nonFunctionalProperties
    dc:description hasValue "A request to add items to a cart"
  endNonFunctionalProperties
  cartId ofType (1) _string
  hmac ofType (1) _string
  mergeCart ofType (1) _string
  items ofType (1 *) item
  responseGroup ofType _string

```

## Cart Add Response

The cart add response simply defines the cart with the new items added. This cart is returned by the *CartAdd* operation. The equivalent concept in the ontology is the cart itself.

Listing 45. CartAddResponse Type

```

<xs:element name="CartAddResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Cart" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Cart Create Request

Defines the information needed to create a cart object. Instances of this datatype are used as input in the *CartCreate* operation.

Listing 46. CartCreateRequest Type

```
<xs:complexType name="CartCreateRequest">
  <xs:sequence>
    <xs:element name="MergeCart" type="xs:string" minOccurs="0"/>
    <xs:element name="Items" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ASIN" type="xs:string" minOccurs="0"/>
                <xs:element name="OfferListingId" type="xs:string"
                  minOccurs="0"/>
                <xs:element name="Quantity" type="xs:positiveInteger"
                  minOccurs="0"/>
                <xs:element name="AssociateTag" type="xs:string"
                  minOccurs="0"/>
                <xs:element name="ListItemId" type="xs:string" minOccurs="0"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The equivalent concept is defined in Listing 47. Note that the attribute "items" of this concept is of type "minimalItem" such that the minimal amount of information is sent from the client to the server.

Listing 47. cartCreateRequest concept

```
concept cartCreateRequest
nonFunctionalProperties
  dc:description hasValue "A request to create a cart object"
endNonFunctionalProperties
mergeCart ofType (1) _string
items ofType minimalItem
responseGroup ofType _string
```

## Cart Create Response

Instances of this concept are returned by the *CartCreate* operation. Similarly for other operations, a cart concept is mapped to this datatype.

Listing 48. CartCreateResponse Type

```
<xs:element name="CartCreateResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Cart" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Cart Modify Request

This request defines the required information to allow the modification of a cart object. The element "action" allows to specify what kind of modification the client will allow. This modification can either be "MoveToCart" or "SaveForLater". The datatype is specified as in Listing 49.

Listing 49. CartModifyRequest Type

```
<xs:complexType name="CartModifyRequest">
  <xs:sequence>
    <xs:element name="CartId" type="xs:string" minOccurs="0"/>
    <xs:element name="HMAC" type="xs:string" minOccurs="0"/>
    <xs:element name="MergeCart" type="xs:string" minOccurs="0"/>
    <xs:element name="Items" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Action" minOccurs="0">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="MoveToCart"/>
                      <xs:enumeration value="SaveForLater"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="CartItemid" type="xs:string" minOccurs="0"/>
                <xs:element name="Quantity" type="xs:nonNegativeInteger"
                  minOccurs="0"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The "cartModifyRequest" concept and the "actionType" axiom definitions are shown in Listing 50.

```

concept cartModifyRequest
  nonFunctionalProperties
    dc#description hasValue "Defines the required constraints to allow
      modification of a cart"
    dc#relation hasValue actionType
  endNonFunctionalProperties
  cartId ofType (1) _string
  hmac ofType (1) _string
  mergeCart ofType (1) _string
  items ofType minimalItem
  action ofType (1) _string
  responseGroup ofType _string

axiom actionType
  nonFunctionalProperties
    dc#description ofType "Defines the types of actions to allow
      modification of a cart"
  endNonFunctionalProperties
  definedBy !-
    ?x memberof cartModifyRequest and
    ?x[action ofType ?actionType] and
    (
      ?actionType hasValue "MoveToCart" or
      ?actionType hasValue "SaveForLater"
    ).

```

## Cart Modify Response

This message is very simple and returns a cart object as the result of the modification. Hence, the equivalent concept for this response is simply a cart object. The datatype is defined in Listing 51.

Listing 51. CartModifyResponse Type

```

<xs:element name="CartModifyResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Cart" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Cart Clear Request

Defines the required constraints to clear items from an existing cart. Instances of this type are used as input for the *CartClear* operation.

Listing 52. CartClearRequest Type

```

<xs:complexType name="CartClearRequest">
  <xs:sequence>
    <xs:element name="CartId" type="xs:string" minOccurs="0"/>
    <xs:element name="HMAC" type="xs:string" minOccurs="0"/>
    <xs:element name="MergeCart" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent "cartClearRequest" concept is specified in Listing 53 below.

Listing 53. cartClearRequest concept

```

concept cartClearRequest
  nonFunctionalProperties
    dc:description hasValue "Request to clear the cart"
  endNonFunctionalProperties
  cartId ofType (1) _string
  hmac ofType (1) _string
  mergeCart ofType (1) _string
  responseGroup ofType _string

```

## Cart Clear Response

This datatype simply specifies the empty cart as a result of the *CartClear* operation. Thus, an instance of "cart" is returned.

Listing 54. CartClearResponse Type

```

<xs:element name="CartClearResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Cart" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

## Transaction Lookup Request

Defines the required constraints required to search for a transaction using the *TransactionLookup* operation.

Listing 55. TransactionLookupRequest Type

```

<xs:complexType name="TransactionLookupRequest">
  <xs:sequence>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="TransactionId" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent "transactionLookupRequest" concept is shown in Listing 56.

Listing 56. transactionLookupRequest concept

```

concept transactionLookupRequest
  nonFunctionalProperties
    dc:description hasValue "Specifies information to search for a
      specific transaction"
  endNonFunctionalProperties
  responseGroup ofType _string
  transactionId ofType (1) _string

```

## Transaction Lookup Response

Instances of this type are returned by the *TransactionLookup* operation and it simply defines a list of transactions that match the input criteria of the operation as shown in Listing 57.

```

<xs:element name="TransactionLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:Transactions" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Transactions">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element name="TotalResults" type="xs:nonNegativeInteger" minOccurs="0"/>
      <xs:element name="TotalPages" type="xs:nonNegativeInteger" minOccurs="0"/>
      <xs:element ref="tns:Transaction" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Transaction">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TransactionId" type="xs:string"/>
      <xs:element name="SellerId" type="xs:string"/>
      <xs:element name="Condition" type="xs:string"/>
      <xs:element name="TransactionDate" type="xs:string"/>
      <xs:element name="TransactionDateEpoch" type="xs:string"/>
      <xs:element name="SellerName" type="xs:string" minOccurs="0"/>
      <xs:element name="PayingCustomerId" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:appinfo>
            <aws-se:restricted
              xmlns:aws-se="http://webservices.amazon.com/AWS-SchemaExtensions">
              <aws-se:excludeFrom>public</aws-se:excludeFrom>
              <aws-se:excludeFrom>partner</aws-se:excludeFrom>
            </aws-se:restricted>
          </xs:appinfo>
        </xs:annotation>
        <xs:element name="OrderingCustomerId"
          type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:appinfo>
              <aws-se:restricted
                xmlns:aws-se="http://webservices.amazon.com/AWS-SchemaExtensions">
                <aws-se:excludeFrom>public</aws-se:excludeFrom>
                <aws-se:excludeFrom>partner</aws-se:excludeFrom>
              </aws-se:restricted>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:element>
      <xs:element name="Totals" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Total" type="tns:Price"/>
            <xs:element name="Subtotal" type="tns:Price"/>
            <xs:element name="Tax" type="tns:Price"/>
            <xs:element name="ShippingCharge" type="tns:Price"/>
            <xs:element name="Promotion" type="tns:Price"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="TransactionItems" minOccurs="0">
        <xs:annotation>

```

The equivalent concept required takes a much simpler form. In principle, the "transaction" concept should be specified by a commerce ontology and the container specified in the amazon ontology as shown in Listing 58.

Listing 58. transactionContainer concept

```

concept transactionContainer
nonFunctionalProperties
  dc:description hasValue "A container for transactions"
endNonFunctionalProperties
  transactions ofType commerce#transaction
  
```

## Seller Listing Search Request

The *SellerListingSearch* operation accepts as input instances of this datatype. It allows to search for zShopes and Marketplace listings and as shown in Listing 59.

Listing 59. SellerListingSearchRequest Datatype

```

<xs:complexType name="SellerListingSearchRequest">
  <xs:sequence>
    <xs:element name="BrowseNode" type="xs:string" minOccurs="0"/>
    <xs:element name="Country" type="xs:string" minOccurs="0"/>
    <xs:element name="Keywords" type="xs:string" minOccurs="0"/>
    <xs:element name="ListingPage" type="xs:positiveInteger" minOccurs="0"/>
    <xs:element name="OfferStatus" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Open"/>
          <xs:enumeration value="Closed"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="PostalCode" type="xs:string" minOccurs="0"/>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="SearchIndex" type="xs:string" minOccurs="0"/>
    <xs:element name="SellerId" type="xs:string" minOccurs="0"/>
    <xs:element name="ShipOption" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ShipTo"/>
          <xs:enumeration value="ShipFrom"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="Sort" type="xs:string" minOccurs="0"/>
    <xs:element name="Title" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
  
```

The equivalent concept "sellerListingSearchRequest" is shown in Listing 60. Two axioms constraining the values offer status and shipping option are also defined

```

concept sellerListingSearchRequest
  nonFunctionalProperties
    dc#description hasValue "Allows to specify constraints to
      search for zShops and Marketplaces"
    dc#relation hasValue {offerStatusType, shipCriteria}
  endNonFunctionalProperties
  browseNode ofType (1) _string
  country ofType (1) _string
  keywords ofType _string
  offerStatus ofType (1) _string
  postalCode ofType (1) _string
  responseGroup ofType _string
  sellerId ofType (1) _string
  shipOption ofType (1) _string
  sort ofType (1) _string
  title ofType (1) _string

axiom offerStatusType
  nonFunctionalProperties
    dc#description hasValue "An offer status can be either
      'Open' or 'Closed'"
  endNonFunctionalProperties
  definedBy !-
    ?x memberOf sellerListingSearchRequest and
    ?x[offerStatus hasValue ?status] and
    (
      ?status hasValue "Open" or
      ?status hasValue "Closed"
    ).

axiom shipCriteria
  nonFunctionalProperties
    dc#description hasValue "A shipping option can be either
      'ShipTo' or 'ShipFrom'"
  endNonFunctionalProperties
  definedBy !-
    ?x memberOf sellerListingSearchRequest and
    ?x[shipOption hasValue ?option] and
    (
      ?option hasValue "ShipTo" or
      ?option hasValue "ShipFrom"
    ).

```

## Seller Listing Search Response

Instances of this type are returned by the *SellerListingSearch* operation. It encloses a set of seller listing instances which match the input criteria specified by a seller listing search request instance. The relevant types are shown in Listing 61 below.

```

<xs:element name="SellerListingSearchResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>
      <xs:element ref="tns:SellerListings" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="SellerListings">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:Request" minOccurs="0"/>
      <xs:element name="TotalResults" type="xs:nonNegativeInteger" minOccurs="0"/>
      <xs:element name="TotalPages" type="xs:nonNegativeInteger" minOccurs="0"/>
      <xs:element ref="tns:SellerListing" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="SellerListing">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ExchangeId" type="xs:string" minOccurs="0"/>
      <xs:element name="ListingId" type="xs:string" minOccurs="0"/>
      <xs:element name="ASIN" type="xs:string" minOccurs="0"/>
      <xs:element name="Title" type="xs:string" minOccurs="0"/>
      <xs:element name="Price" type="tns:Price" minOccurs="0"/>
      <xs:element name="StartDate" type="xs:string" minOccurs="0"/>
      <xs:element name="EndDate" type="xs:string" minOccurs="0"/>
      <xs:element name="Status" type="xs:string" minOccurs="0"/>
      <xs:element name="Quantity" type="xs:string" minOccurs="0"/>
      <xs:element name="QuantityAllocated" type="xs:string" minOccurs="0"/>
      <xs:element name="Condition" type="xs:string" minOccurs="0"/>
      <xs:element name="SubCondition" type="xs:string" minOccurs="0"/>
      <xs:element name="ConditionNote" type="xs:string" minOccurs="0"/>
      <xs:element name="Availability" type="xs:string" minOccurs="0"/>
      <xs:element name="FeaturedCategory" type="xs:string" minOccurs="0"/>
      <xs:element ref="tns:Seller" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The "sellerListingContainer" and "sellerListing" concepts are defined in Listing 62.

```

concept sellerListingContainer
  nonFunctionalProperties
    dc#description hasValue "A container for Seller Listings"
  endNonFunctionalProperties
  sellerListings ofType sellerListing

concept sellerListing
  nonFunctionalProperties
    dc#description hasValue "A seller listing"
  endNonFunctionalProperties
  exchangeId ofType (1) _string
  listingId ofType (1) _string
  asin ofType (1) _string
  title ofType (1) _string
  price ofType (1) commerce#price
  startDate ofType (1) _string
  endDate ofType (1) _string
  status ofType (1) _string
  quantity ofType (1) _string
  quantityAllocated ofType (1) _string
  availability ofType _string
  featuredCategory ofType _string
  seller ofType commerce#seller

```

## Seller Listing Lookup Request

Allows to search for a zShop or Marketplace using an identifier. Instances of this type are accepted as input by the *SellerListingLookup* operation.

Listing 63. TransactionLookupResponse, Transactions and Transaction Types

```

<xs:complexType name="SellerListingLookupRequest">
  <xs:sequence>
    <xs:element name="Id" type="xs:string" minOccurs="0"/>
    <xs:element name="IdType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Exchange"/>
          <xs:enumeration value="Listing"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

The equivalent "sellerListingLookupRequest" concept is shown in Listing 64. An axiom restricting the type of identifier is also specified.

Listing 64. transactionLookupRequest concept

```

concept sellerListingLookupRequest
  nonFunctionalProperties
    dc#description hasValue "Allows to specify constraints to search
      for zShops and Marketplaces using an identifier"
    dc#relation hasValue {sellerListingIdType}
  endNonFunctionalProperties
  id ofType (1) _string
  idType ofType (1) _string

```

```
responseGroup ofType _string
```

```
axiom sellerListingIdType
```

```
nonFunctionalProperties
```

```
dc:description hasValue "An identifier type for seller listings  
can be 'Exchange' or 'Listing'"
```

```
endNonFunctionalProperties
```

```
definedBy !-
```

```
?x memberof sellerListingLookupRequest and
```

```
?x[idType hasValue ?type] and
```

```
(  
  ?type hasValue "Exchange" or  
  ?type hasValue "Listing"
```

```
).
```

## Seller Listing Lookup Response

Similarly for *SellerListingSearchResponse*, this datatype is used as a container for defining the Listings that match the input criterion of the *SellerListingLookup* operation. Therefore, the "sellerListingContainer" defined in Listing 62 is used to map to this type.

Listing 65. SellerListingLookupResponse Type

```
<xs:element name="SellerListingLookupResponse">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element ref="tns:OperationRequest" minOccurs="0"/>  
      <xs:element ref="tns:SellerListings" minOccurs="0" maxOccurs="unbounded"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

## 4. WSMO Capability of the Amazon E-commerce Service

As we can see in the previous section, the functionality of the ECS consists of five independent parts - product data inquiry, shopping cart manipulation, customer content inquiry, seller information inquiry and several other operations. While we could model the ECS in WSMO as five Web services (one for each group of operations, described in detail above in [Section 2](#)), we follow Amazon's decision to put them all in a single big Web service description.

The overall WSMO capability of ECS is that the client can get several types of information and/or that the client can create and manipulate the content of shopping carts. In particular, the capability is as follows: the precondition and output of the ECS is that the client has information about Amazon products and/or a shopping cart ready for manual completion of the purchase. The precondition and input is that the client wants to get information about Amazon products or that the client wants to purchase some products (and has some data that distinguishes the products).

The capability of the WSMO Web service for Amazon is described in the following sub-sections. The namespaces and non-functional properties for the web service are first defined (in Section 4.1). The pre- and post-conditions are then described separately in Section 4.2, and respectively Section 4.3. Each post-condition is described separately since the description is quite exhaustive.

### 4.1 WSMO Web Service Definition for Amazon E-commerce Service

Listing 66 describes the namespace and non-functional properties declarations for the WSMO Amazon Web service. Note that the commerce ontology is not yet defined and we will thus use a fictitious ontology. We will assume that the capability described in the next sections is in the same namespace.

```

namespace { _"http://www.wsmo.org/webServices/amazonWS#",
  dc _"http://purl.org/dc/elements/1.1#",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  am _"http://www.wsmo.org/ontologies/amazon/amazonOntology#",
  bk _"http://www.wsmo.org/ontologies/amazon/bookOntology#",
  commerce _"http://example.org/commerce",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}

webService _"http://www.wsmo.org/webServices/amazonWS"
nonFunctionalProperties
  dc#title hasValue "WSMO for Amazon"
  dc#creator hasValue "James Scicluna"
  dc#subject hasValue {"Book", "Book Searching", "Book Buying"}
  dc#description hasValue "WSMO Web service for Book searching and ordering for amazon"
  dc#publisher hasValue "DERI Innsbruck"
  dc#contributor hasValue {"Jacek Kopecky", "Dumitru Roman",
    "Axel Polleres", "Jos de Bruijn"}
  dc#date hasValue "2005-06-30"
  dc#type hasValue _"http://www.wsmo.org/2004/d2#webservice"
  dc#format hasValue "text/html"
  dc#identifier hasValue _"http://www.wsmo.org/webServices/amazonWS"
  dc#source hasValue _"http://www.wsmo.org/webServices/amazonWS"
  dc#language hasValue "en-US"
  dc#relation hasValue {
    _"http://www.wsmo.org/ontologies/amazon/amazonOntology",
    _"http://soap.amazon.com/schemas2/AmazonWebServices.wsdl"
  }
  dc#coverage hasValue "ID:7029392 Name:World"
  wsml#version hasValue "$Revision: 1.5 $"
  wsml#endpointDescription hasValue
  _"http://webservices.amazon.com/AWSECommerceService/2005-03-23#wsdl.service(AWSECommerceS
endNonFunctionalProperties

```

## 4.2 Pre-Conditions

All inputs of the Amazon web service are captured by the preconditions. Amazon can accept *any* of the inputs in a given moment and hence the precondition is defined in terms of a disjunction of all the possible inputs as shown in Listing 67. The only shared variable is *?request* which is required in order to capture what post-condition will take effect given a particular input.

```

capability amazonWSCapability
nonFunctionalProperties
  dc:description hasValue "This capability provides a comprehensive
    description of all the Amazon Web service. Hence, all inputs and
    outputs of the Amazon operations are captured in this description"
endNonFunctionalProperties

sharedVariables {?request}

precondition
nonFunctionalProperties
  dc:description hasValue "The Amazon service handles requests
    for:
    - Help
    - Searching for items by keywords
    - Lookup items by some identifier
    - Browsing Nodes
    - List searching by keywords
    - List lookup by an identifier
    - Customer Content Search by keywords
    - Customer Content Search by an identifier
    - Searching for similar items by an identifier
    - Seller lookup by an identifier
    - Get a cart object by an identifier
    - Add items to a cart
    - Creation of a cart
    - Modification of a cart
    - Clear a cart
    - Lookup for Transaction by an identifier
    - Search for a Seller Listing by keywords
    - Lookup for a Seller Listing by an identifier"
endNonFunctionalProperties
definedBy
  //Request for a help topic
  ?request memberOf am#helpRequest or
  //A search by author, keywords, title, minPrice or maxPrice
  ?request memberOf am#itemSearchRequest or
  //A search by ID
  ?request memberOf am#itemLookupRequest or
  //Browse a particular node
  ?request memberOf am#browseNodeLookupRequest or
  //Search for a list
  ?request memberOf am#listSearchRequest or
  //Search for a list by an identifier
  ?request memberOf am#listLookupRequest or
  //Search for a customer by keywords
  ?request memberOf am#customerSearchRequest or
  //Search for a customer by an identifier
  ?request memberOf am#customerLookupRequest or
  //Search for similar items
  ?request memberOf am#similarityLookupRequest or
  //Search for a seller by an identifier
  ?request memberOf am#sellerLookupRequest or
  //A request to create a cart
  ?request memberOf am#cartCreateRequest or
  //A request to get the cart
  ?request memberOf am#cartGetRequest or
  //A request to add items to a cart
  ?request memberOf comr#cartAddRequest or
  //Request to modify a cart
  ?request memberOf am#cartModifyRequest or
  //A request to clear the cart
  ?request memberOf comr#cartClearRequest.

```

## 4.3 Post-Conditions

The post-condition take the form of implication rules. That is, given a particular pre-condition, one particular post-condition will take effect. The general definition of the postcondition is given in Listing 68; Listings from 69 to 86 describe the particular postcondition of each operation, which are part of the general postcondition of the WSMO Web service.

Listing 68. Post-Condition definition.

```
postcondition  
nonFunctionalProperties  
  dc#description hasValue "The Service can return:  
    - a list of searched items (either by ID, keywords or similarity)  
    - a cart  
    - customer information  
    - seller information  
    - browse nodes  
    - lists  
    - transactions"  
endNonFunctionalProperties
```

### Help Request

Given a request for some particular type of help, a response with the same type is returned. In Amazon, the type of help can be either for an operation or a response group. Listing 69 defines this simple post-condition.

Listing 69. Post-Condition for Help Request.

```
postcondition  
nonFunctionalProperties  
  
  dc#description hasValue "The Service can return:  
    - a list of searched items (either by ID, keywords or similarity)  
    - a cart  
    - customer information  
    - seller information  
    - browse nodes  
    - lists  
    - transactions"  
endNonFunctionalProperties  
definedBy  
  // Given a request for some particular type of help, a response with the same  
    // type is returned. In Amazon, the type of help can be either for an operation  
    // or a response group.  
    (?request[helpType hasValue ?type] memberOf  
      am#helpRequest implies  
        exists ?response (?response[responseType hasValue ?type]  
          memberOf am#helpResponse)  
    ) and
```

### Search for Items by Keywords

The post-condition for searching by keywords is more complex. The client must define *at least* one keyword and a container with a set of items which match the keyword(s) is returned. The criterion

include: author, keywords, title and minimum and maximum price as defined in Listing 70.

Listing 70. Post-Condition for Search for an Item by keywords.

```
// The result for a search request by author is a container of products
// such that all items in the container are amazonBooks have the requested author:
(?request[author hasValue ?author] memberOf am#searchBooks implies
exists ?container (?container memberOf am#itemContainer and
forall ?item (?container[items hasValue ?item]
implies ?item[author hasValue ?author] memberOf bk#amazonBook))
) and
// The result for a search request by keyword is a container of products
// such that all items in the container are amazonBooks have the requested keyword:
(?request[keywords hasValue ?keyword] memberOf am#searchBooks implies
exists ?container (?container memberOf am#itemContainer and
forall ?item (?container[items hasValue ?item] implies
?item[keywords hasValue ?keyword] memberOf bk#amazonBook))
) and
// The result for a search request by title is a container of products
// such that all items in the container have the requested title (not necessarily
// only books):
(?request[title hasValue ?title] memberOf am#searchBooks implies
exists ?container (?container memberOf am#itemContainer and
forall ?item (?container[items hasValue ?item] implies
?item[title hasValue ?title])
)) and
// The result for a search request by minPrice is a container of products
// such that all items in the container have a price >= than the requested
// minPrice (not necessarily only books):
(?request[minPrice hasValue ?minPrice] memberOf am#searchBooks implies
exists ?container (?container memberOf am#itemContainer and
forall{?item,?price} (?container[items hasValue ?item] and
?item[price hasValue ?price] implies
?price >= ?minPrice))) and
// The result for a search request by maxPrice is a container of products
// such that all items in the container have a price <= than the
// requested maxPrice (not necessarily only books):
(?request[maxPrice hasValue ?maxPrice] memberOf am#searchBooks implies
exists ?container (?container memberOf am#itemContainer and
forall {?item,?price} (?container[items hasValue ?item] and
?item[price hasValue ?price] implies
?price <= ?maxPrice))) and
```

## Search for Items by an Identifier

Similarly for the search by keywords, the result of such a search request is an item container which holds the item(s) which match the particular identifier defined by the client.

Listing 71. Post-Condition for Search for Items by an Identifier.

```
// The result for a search request by id is a container of products
// such that all items in the container have the requested id:
(?request[id hasValue ?id] memberOf requests#searchById implies
exists ?container (?container memberOf am#itemContainer and
forall {?item} (?container[items hasValue ?item] implies
?item[id hasValue ?id]))) and
```

## Creation of a Cart

Given a set of items, a cart which holds these items is returned back to the client as described in Listing 72.

Listing 72. Post-Condition for Search for Items by an Identifier.

```
// How can I express now "with a new ID"?  
(?request[items hasValue ?cartItem] memberOf am#createCart implies  
  exists ?cart (?cart[items hasValue ?cartItem] memberOf am#cart)  
  ) and
```

## Getting a Cart

Given the identifier of a cart, the respective cart object is to be returned. Listing 73 defines such a post-condition

Listing 73. Post-Condition for Search for Items by an Identifier.

```
// Very similar to CartCreate  
(?request[id hasValue ?id] memberOf am#cartGetRequest implies  
  exists ?cart (?cart[id hasValue ?id] memberOf am#cart)  
  ) and
```

## Browsing a Node

The products of Amazon are listed into groups which the client can browse. Examples of such groups include books, tools, music and others. The groups are categorized into hierarchies which are called browse areas. Each group has then a specific node identifier in the particular browse area. Amazon allows to ask for a browse node which also defines its child nodes that can be browsed. The postcondition for such an operation is defined in Listing 74.

Listing 74. Post-Condition for Browsing a Node.

```
// The products of Amazon are listed into groups which the client can  
  
// browse. Examples of such groups include books, tools, music and others. The  
  
// groups are categorized into hierarchies which are called browse areas. Each  
  
// group has then a specific node identifier in the particular browse area.  
  
// Amazon allows to ask for a browse node which also defines its child nodes  
  
// that can be browsed.  
(?request[nodeId hasValue ?id] memberOf am#browseNodeLookupRequest implies  
  exists ?node (?node[nodeId hasValue ?id] memberOf am#browseNode) and  
  forall ?nodeResponse (?nodeResponse[nodes hasValue ?node]  
    memberOf am#browseNodeLookupResponse)  
  ) and
```

## Searching for a List by Keywords

It is possible to search in Amazon for a wish list, baby list or a wedding registry. This can be done either by keywords (as in this case) or by an identifier. The postcondition for searching by keywords is defined in Listing 75.

```
// PostCondition for Searching for a list by keywords. The list registries include wish-lists, baby-lists and
(?request[listType hasValue ?type] memberOf am#listSearchRequest implies
exists ?someList (?someList[listType hasValue ?type] memberOf am#list) and
forall ?listResponse (?listResponse[lists hasValue ?someList] memberOf am#listContainer)
) and
```

### Searching for a List by an Identifier

This is similar to the case above but allows only to search by an identifier.

Listing 76. Post-Condition for Searching for a List by an Identifier.

```
// Postcondition for searching for a list by an identifier
(?request[listId hasValue ?id] memberOf am#listLookupRequest implies
exists ?someList (?someList[listId hasValue ?id] memberOf am#list) and
forall ?listResponse (?listResponse[lists hasValue ?someList]
memberOf am#listContainer)
) and
```

### Searching for a Customer by keywords

Amazon allows to search for a customer using the name and email address. The relevant customers are returned only if they have made this information public. The postcondition of such an operation is defined in Listing 77 below.

Listing 77. Post-Condition for Searching for a Customer by email and name

```
// PostCondition for Customer Content search by keywords
(?request[name hasValue ?cName, email hasValue ?cEmail]
memberOf am#customerSearchRequest implies
exists ?customer (?customer[name hasValue ?cName, email hasValue ?cEmail]
memberOf commerce#customer) and
forall ?response (?response[customers hasValue ?customer] memberOf am#customerContainer)
) and
```

### Searching for a Customer by an identifier

Similar for above but allows to search for a customer using the identifier as described in Listing 78.

Listing 78. Post-Condition for Searching for a Customer by an Identifier

```
// PostCondition for Customer Content search by an identifier
(?request[customerId hasValue ?id] memberOf am#customerLookupRequest
implies
exists ?customer (?customer[customerId hasValue ?id]
memberOf commerce#customer) and
forall ?response (?response[customers hasValue ?customer]
memberOf am#customerContainer)
) and
```

## Searching for similar products

It is allowed to search for some similar products in amazon. The postcondition of such an operation is defined in Listing 79.

Listing 79. Post-Condition for Searching for Similar Products

```
// PostCondition for searching for similar products
// There's a problem here: There's no attribute that can connect the
// input with the input
(?request[itemId hasValue ?sourceId] memberOf am#similarityLookupRequest
implies
exists {?sourceItem, ?targetItem} (?sourceItem[itemId hasValue ?sourceId]memberOf am#item a
?targetItem memberOf am#item and am#similarItems(?sourceItem, ?targetItem)) and
forall ?response (?response[items hasValue ?targetItem] memberOf am#itemContainer)
) and
```

## Searching for a Seller by an Identifier

Information about a seller can be looked up using an identifier. The post-condition of this operation is defined in Listing 80 below.

Listing 80. Post-Condition for Searching for a seller using an Identifier

```
// PostCondition for searching for a Seller by an identifier
(?request[sellerId hasValue ?id] memberOf am#sellerLookupRequest implies
exists ?seller (?seller[sellerId hasValue ?id] memberOf commerce#seller) and
forall ?response (?response[sellers hasValue ?seller] memberOf am#sellerContainer)
) and
```

## Adding Items to a Cart

New items can be added to cart objects. The cart identifier and the new items to add must be specified and the cart object with the added items is returned. The post-condition for this operation is described in Listing 81.

Listing 81. Post-Condition for adding items to a Cart

```
// A request to add items to a cart.
(?request[id hasValue ?id, item hasValue ?item] memberOf comr#addToCart and
?cart[id hasValue ?id] memberOf am#cart implies
?cart[item hasValue ?item]
) and
```

## Modifying the Cart

Amazon allows to modify the quantity of items in a cart as defined in the post-condition in Listing 82 below.

Listing 82. Post-Condition for modifying a Cart

```
// PostCondition for modifying a cart. It is similar to the addToCart request
(?request[cartId hasValue ?id] memberOf am#cartModifyRequest implies
exists ?cart (?cart[id hasValue ?id] memberOf am#cart)
) and
```

### Clear the contents of a Cart

This operation allows to clear all the items in a cart. A cart identifier is received by the web service and the same empty cart is returned. The post-condition of this operation is defined in Listing 83 below:

Listing 83. Post-Condition for clearing items from a Cart

```
// Postcondition for clearing the contents of a cart:
(?request[id hasValue ?id] memberOf comr#cartClearRequest and
?cart[id hasValue ?id] memberOf am#cart implies
neg exists ?item (?cart[item hasValue ?item])
) and
```

### Searching for a Transaction by an identifier

Amazon allows to search for transactions using only specific identifiers. The post-condition of such an operation is defined in Listing 84 below:

Listing 84. Post-Condition for Searching for a Transaction by an identifier

```
// PostCondition for searching for a transaction by an identifier
(?request[transactionId hasValue ?id] memberOf am#transactionLookupRequest implies
exists ?transaction (?transaction[transactionId hasValue ?id]
memberOf commerce#transaction) and
forall ?response (?response[transactions hasValue ?transaction]
memberOf am#transactionContainer)
) and
```

### Searching for a Seller Listing

It is possible to search for zShops and marketplaces using this operation. The post-condition is described in Listing 85.

Listing 85. Post-Condition for Searching for zShops or marketplaces

```
// PostCondition for searching for a Seller Listing
(?request[sellerId ofType ?id] memberOf am#sellerListingSearchRequest implies
exists {?seller, ?listing, ?listSeller}
(?seller[sellerId hasValue ?id] memberOf commerce#seller and
?listing[seller hasValue ?listSeller] memberOf am#listing and
?listSeller[sellerId hasValue ?id] memberOf commerce#seller) and
forall ?response (?response[sellerListings hasValue ?listing]
memberOf am#sellerListingContainer)
) and
```

## Searching for a Seller Listing using an Identifier

Similar for the post-condition above but using an identifier as defined in Listing 86.

Listing 86. Post-Condition for Searching for zShops or marketplaces using an Identifier

```
// PostCondition for searching for a Seller Listing using a type of listing
// NOTE: Not sure if the modelling below reflects what Amazon does
(?request[id hasValue ?listingType] memberOf am#sellerListingLookupRequest implies
exists {?seller, ?listing}
  (?seller[exchangeId hasValue ?listingType] memberOf commerce#seller and
  ?listing[seller hasValue ?seller] memberOf am#sellerListing) and
forall ?response (?response[sellerListings hasValue ?sellerListing]
  memberOf am#sellerListingContainer)
).
```

## 5. WSMO Choreography for Amazon E-commerce Service

This section describes the interface part of the amazon service, beginning with the state signature (together with the input and output modes) and following with the transition rules of the choreography interface. Again, we assume that the listings presented here are in the same namespace as for the capability.

### 5.1 State Signature

The state signature of the choreography first imports the amazon ontology which defines the state of the abstract state machine. The modes *in* and *out* are then defined. The required concepts are directly mapped to the input message of some operation (for the *in* modes) and to the output message (for the *out* modes). The state signature of the choreography interface is described in Listing 87 below.

```

interface amazonWSInterface
choreography
stateSignature
  importsOntology _ "http://www.wsmo.org/ontologies/amazon/amazonOntology"

in
  concept am#itemSearchRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/ItemSearch/In)"
  concept am#itemLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/ItemLookup/In)"
  concept am#browseNodeLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/BrowseNodeLookup/In)"
  concept am#listSearchRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/ListSearch/In)"
  concept am#listLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/ListLookup/In)"
  concept am#customerSearchRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CustomerContentSearch/In)"
  concept am#customerLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CustomerContentLookup/In)"
  concept am#similarityLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/SimilarityLookup/In)"
  concept am#sellerLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/SellerLookup/In)"
  concept am#cartCreateRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CartCreate/In)"
  concept am#cartGetRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CartGet/In)"
  concept am#cartAddRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CartAdd/In)"
  concept am#cartModifyRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CartModify/In)"
  concept am#cartClearRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/CartClear/In)"
  concept am#transactionLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/TransactionLookup/In)"
  concept am#sellerListingSearchRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/SellerListingSearch/In)"
  concept am#sellerListingLookupRequest withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/SellerListingLookup/In)"

out
  concept am#helpResponse withGrounding
    _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
      wsdl.interfaceMessageReference(AWSECommerceServicePortType/Help/Out)"

  concept am#browseNodeLookupResponse withGrounding

```

## 5.2 Transition Rules

For the choreography of the Amazon E-Commerce Service, we model a single transition rule for each of the service's operations. Each operation is described below together with its particular transition rule.

In general, the transition rules only check for the presence of a request (whose validity should be assured by the ontology definition) and add the respective result in response, but several rules (in particular, for the operations `CartAdd`, `CartGet`, `CartModify`, `CartClear`, and all the `*Lookup` operations) also check for the existence of the appropriate cart or the instance that's the object of the lookup operation. This implies that the client either has to know of an existing cart or it has to call `CartCreate` before all the other cart manipulation operations, and that if the client doesn't know of an item otherwise, it can't call `ItemLookup`, for example. In effect, this is an example of implicit control over actions.

These conditions are only implied in the ECS documentation by the fact that these parameters are required for some operations (e.g. `cartID` for `CartAdd`) and are produced by other operations (`CartCreate`).

### Product Data Operations

Product data operations allow you to look up Amazon products, either by searching for groups of products or by looking up specific items.

**"ItemSearch" operation** - searches the product database using a one of a variety of possible criteria.

Listing 88. Transition Rule for the "ItemSearch" operation.

```
if (?ItemSearchRequest memberOf am#itemSearchRequest) then
  add(_# memberOf am#itemSearchResponse)
endif
```

**"ItemLookup" operation** - looks up specific product(s) in the Amazon database.

Listing 89. Transition Rule for the "ItemLookup" operation.

```
if(?ItemLookupRequest[
  itemId hasValue ?itemId
]memberOf am#itemLookupRequest) and
(exists ?item
 (?item[
   asin hasValue ?itemId
 ]memberOf am#item
 )
) then
  add(_# memberOf am#itemLookupResponse)
endif
```

**"SimilarityLookup" operation** - looks up products that are similar to specific other products.

Listing 90. Transition Rule for the "SimilarityLookup" operation.

```
if(?SimilarityLookupRequest
  [itemId hasValue ?itemId
  ]memberOf am#SimilarityLookupRequest
) and
(exists ?item
  (?item[
    asin hasValue ?itemId
  ]memberOf am#item
  )
) then
add(_# memberOf am#itemContainer)
endif
```

**"SellerListingSearch" operation** - search for seller product listings (zShops and Marketplace).

Listing 91. Transition Rule for the "SellerListingSearch" operation.

```
if (?SellerListingSearchRequest memberOf am#sellerListingSearchRequest) then
  add(_# memberOf am#sellerListingContainer)
endif
```

**"SellerListingLookup" operation** - looks up a specific zShops or Marketplace product listing from a particular seller.

Listing 92. Transition Rule for the "SellerListingLookup" operation.

```
if(?SellerListingLookupRequest[
  listingId hasValue ?listingId
  ]memberOf am#sellerListingLookupRequest) and
(exists ?listing
  (?listing[
    listingId hasValue ?listingId
  ]memberOf awr#sellerListing
  )
) then
  add(_# memberOf am#sellerListingContainer)
endif
```

## Shopping Cart

The shopping cart operations allow to create a shopping cart of items and hand it off to Amazon so the customer can purchase the selected items.

**"CartCreate" operation** - creates a shopping cart and add an item(s).

Listing 93. Transition Rule for the "CartCreate" operation.

```
if (?CartCreateRequest memberOf am#cartCreateRequest) then
  add(_# memberOf Cart)
endif
```

**"CartAdd" operation** - adds products to the shopping cart. Note that (although not shown here) the adding items to the cart can be considered as non-deterministic since if the quantity of the item requested by the client is more than that available in stock, the items are not added to the cart.

Listing 94. Transition Rule for the "CartAdd" operation.

```
if(?CartAddRequest[
  cartId hasValue ?cartId,
  hmac hasValue ?hmac,
  items hasValue ?newItems
]memberOf am#CartAddRequest) and
(exists ?cart
  (?cart[
    id hasValue ?cartId,
    hmac hasValue ?hmac
  ]memberOf am#Cart
  )then
  add(?cart[items hasValue ?newItems])
  add(_# memberOf am#cart)
endif
```

**"CartModify" operation** - changes quantities or status of entries in the shopping cart.

Listing 95. Transition Rule for the "CartModify" operation.

```
if(?CartModifyRequest[
  cartId hasValue ?cartId,
  hmac hasValue ?hmac,
  items hasValue ?newItems
]memberOf am#CartModifyRequest) and
(exists ?cart
  (?cart[
    id hasValue ?cartId,
    hmac hasValue ?hmac,
    items hasValue ?oldItems
  ]memberOf am#Cart
  )
  ) then
  update(?cart[items hasValue ?oldItems => ?newItems])
  add(_# memberOf CartModifyResponse)
endif
```

**"CartClear" operation** - clears all entries in a shopping cart

Listing 96. Transition Rule for the "CartClear" operation.

```
if(?CartClearRequest[
  cartId hasValue ?cartId,
  hmac hasValue ?hmac
]memberOf am#cartClearRequest) and
(exists ?cart
 (?cart[
  id hasValue ?cartId,
  hmac hasValue ?hmac
 ]memberOf am#cart
 )
) then
add(_# memberOf am#cart)
endif
```

**"CartGet" operation** - gets the contents of a shopping cart with updated price and availability information.

Listing 97. Transition Rule for the "CartGet" operation.

```
if(?CartGetRequest[
  cartId hasValue ?cartId,
  hmac hasValue ?hmac
]memberOf am#cartGetRequest) and
(exists ?cart
 (?cart[
  id hasValue ?cartId,
  hmac hasValue ?hmac
 ]memberOf am#cart
 )
) then
add(_# memberOf am#cart)
endif
```

## Customer Content

Customer content operations allow to look up publicly available customer content, such as reviews, wish lists and Listmania lists.

**"CustomerContentSearch" operation** - searches for customers.

Listing 98. Transition Rule for the "CustomerContentSearch" operation.

```
if (?CustomerContentSearchRequest memberOf am#customerSearchRequest) then
add(_# memberOf am#customerContainer)
endif
```

**"CustomerContentLookup" operation** - looks up specific customer(s) information (location/name/reviews).

Listing 99. Transition Rule for the "CustomerContentLookup" operation.

```
if(?CustomerContentLookupRequest[
```

```

    customerId hasValue ?customerId
]memberOf am#customerLookupRequest
) and
(exists ?customer
  (?customer[
    customerId hasValue ?customerId
    ]memberOf commerce#customer
  )
) then
add(_# memberOf am#customerContainer)
endif

```

**"ListLookup" operation** - looks up specific list(s) created by customers.

Listing 100. Transition Rule for the "ListLookup" operation.

```

if(?ListLookupRequest[
  listId hasValue ?listId
]memberOf am#listLookupRequest) and
(exists ?list
  (?list[
    listId hasValue ?listId
    ]memberOf am#list
  )
) then
add(_# memberOf am#listContainer)
endif

```

**"ListSearch" operation** - searches for customer-created lists.

Listing 101. Transition Rule for the "ListSearch" operation.

```

if (?ListSearchRequest memberOf am#listSearchRequest) then
  add(_# memberOf am#listContainer)
endif

```

## Seller Information

ECS allows you to look up feedback on specific sellers and get seller product listings.

**"SellerLookup" operation** - looks up information about a specific seller.

Listing 102. Transition Rule for the "SellerLookup" operation.

```
if(?SellerLookupRequest[
  sellerId hasValue ?sellerId
]memberOf am#sellerLookupRequest
) and
(exists ?seller
 (?seller[
  sellerId hasValue ?sellerId
 ]memberOf commerce#seller
 )
) then
add(_# memberOf SellerLookupResponse)
endIf
```

## Other Operations

**"BrowseNodeLookup" operation** - returns information about a browse node, including child nodes.

Listing 103. Transition Rule for the "BrowseNodeLookup" operation.

```
if (?BrowseNodeLookupRequest memberOf am#browseNodeLookupRequest) then
  add(_# memberOf am#browseNode)
endIf
```

**"Help" operation** - returns information about how to use an ECS operation or about what to expect from ECS response groups.

Listing 104. Transition Rule for the "Help" operation.

```
if (?HelpRequest memberOf am#helpRequest) then
  add(_# memberOf am#helpResponse)
endIf
```

**"TransactionLookup" operation** - looks up the amount of a specific transaction.

Listing 105. Transition Rule for the "TransactionLookup" operation.

```
if(?TransactionLookupRequest[
  transactionId hasValue ?transactionId
]memberOf am#transactionLookupRequest) and
(exists ?transaction
 (?transaction[
  transactionId hasValue ?transactionId
 ]memberOf commerce#transaction
 )
) then
add(_# memberOf am#transactionContainer)
endIf
```

## 6. Conclusions

This document shows how a WSMO description (including the ontology and Web service elements) is created bottom-up from a WSDL and textual description of the Amazon ECS Web service.

## Appendix A. The Complete WSMO Description for the Amazon E-commerce Service

The complete WSMO Description for the Amazon E-commerce Service can be found in the listing below. The WSML file can be downloaded from [here](#).

```

namespace { _"http://www.wsmo.org/ontologies/amazon/amazonOntology#",
  dc _"http://purl.org/dc/elements/1.1#",
  dt _"http://www.wsmo.org/ontologies/dateTime#",
  loc _"http://www.wsmo.org/ontologies/location#",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  bk _"http://www.wsmo.org/ontologies/amazon/bookOntology#",
  wsm1 _"http://www.wsmo.org/wsm1/wsm1-syntax#",
  graphics _"http://example.org/graphics#",
  commerce _"http://example.org/commerce#",
  ratings _"http://example.org/ratings#",
  music _"http://example.org/music#",
  tasks _"http://example.org/commerceTasks#"
}

ontology _"http://www.wsmo.org/ontologies/amazon/amazonOntology"

nonFunctionalProperties
  dc#title hasValue "Ontology for a WSMO Amazon Web service"
  dc#creator hasValue {"James Scicluna"}
  dc#subject hasValue {"Book Search", "Book Buying"}
  dc#description hasValue "Simple Amazon Ontology for Searching and Buying Books"
  dc#publisher hasValue "DERI Innsbruck"
  dc#contributor hasValue {"Jacek Kopecky", "Dumitru Roman"}
  dc#date hasValue "2005-06-30"
  dc#type hasValue _"http://www.wsmo.org/2004/d2#ontologies"
  dc#format hasValue "text/html"
  dc#identifier hasValue _"http://www.wsmo.org/ontologies/amazon/amazonOntology"
  dc#source hasValue _"http://keg.cs.tsinghua.edu.cn/ontology/travel"
  dc#language hasValue "en-US"
  wsm1#version hasValue "$Revision: 1.5 $"
endNonFunctionalProperties

concept item
  nonFunctionalProperties
    dc#description hasValue "An item handled by the Amazon Service"
  endNonFunctionalProperties
  asin ofType _string
  irl ofType _iri
  salesRank ofType _string
  smallImage ofType graphics#image
  mediumImage ofType graphics#image
  largeImage ofType graphics#image
  offerSummary ofType commerce#offerSummary
  offers ofType commerce#offer
  variationSummary ofType commerce#variationSummary
  variations ofType commerce#variation
  customerReviews ofType ratings#customerReview
  editorialReviews ofType ratings#editorialReview
  similarProduct ofType minimalItem
  accessories ofType minimalItem
  tracks ofType music#track
  promotionalTag ofType _string

concept minimalItem
  nonFunctionalProperties
    dc#description hasValue "Defines the minimal information needed for defining an item"
  endNonFunctionalProperties
  asin ofType _string
  title ofType _string
  quantity ofType (1) _int

concept cartItem
  nonFunctionalProperties
    dc#description hasValue "A cart item is described by a specific item and a quantity"

```

## Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperanto; by Science Foundation Ireland under the DERI-Lion project (Grant No. SFI/02/CE1/I131) and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW<sup>2</sup> and TSC. The editors would like to thank to all the [members of the WSMO working group](#) for their advice and input into this document.

---



\$Date: 2006/01/13 17:48:11 \$

---