VRP-REP TUTORIALS
THE UNIFIED INSTANCE FORMAT

1 Introduction

HERE: short intro on what is xml and xsd

2 Conventions

To illustrate some of the concepts discussed in this tutorial, we have included figures showing tree visualizations of XML Schema elements. We generated these figures using Altova Spy 2014. To help navigate the illustrations Figure 1 presents the conventions used in our illustrations and Figures 2 and 3 present examples of XML Schema tree visualizations.

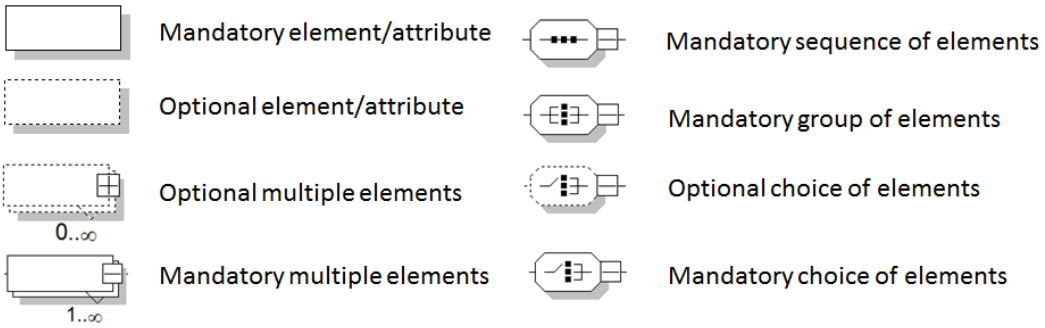

Figure 1: XML Schema tree visualization: conventions

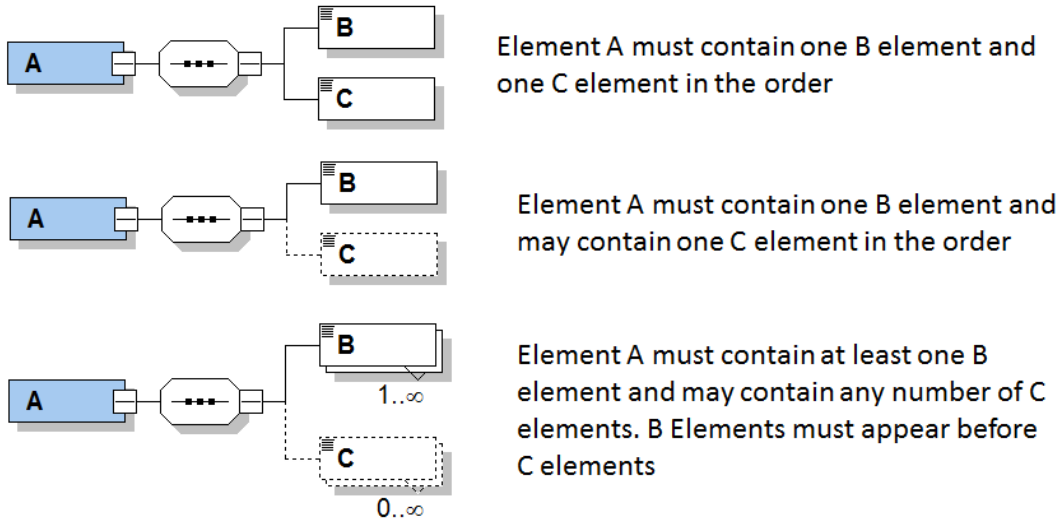

Figure 2: XML Schema tree visualization: example of element sequences

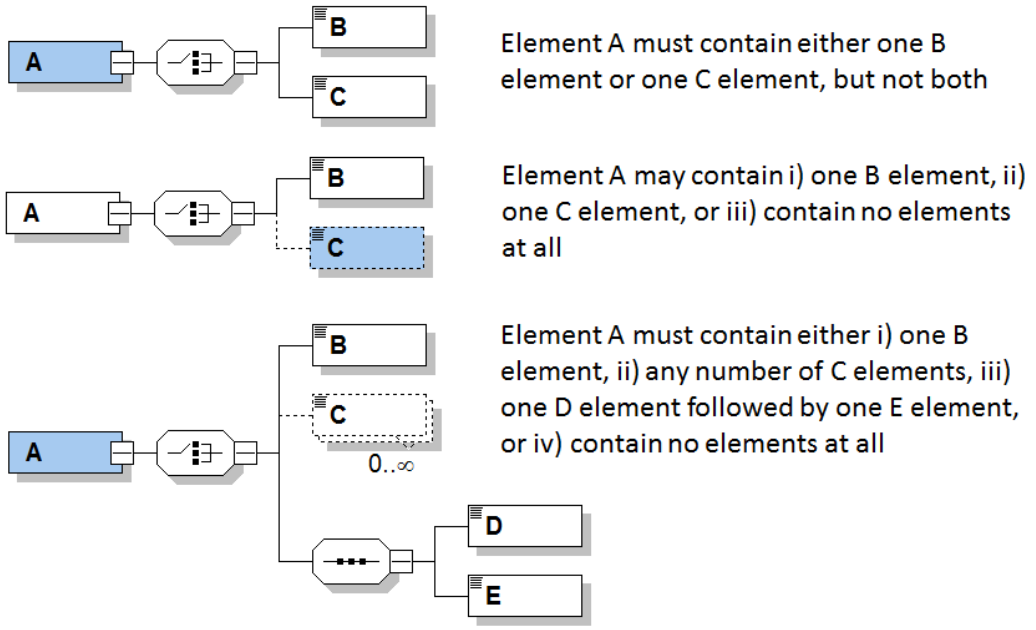

Figure 3: XML Schema tree visualization: example of element choices

3 The unified instance format

3.1 Instance file naming convention

The unique identifier of an instance in the `www.vrp-rep.org` database is the name of its XML instance file (without the .xml extension). Therefore, the name you give to an instance file will become the "official" name of the instance in the repository; it cannot be changed after you upload the instance file.

Good practice. Name your instance file exactly with the same name the instance was given in the article in which it was first introduced.

3.2 Global elements and types

XML schema allows you to define global elements and types that can be re-used in different parts of your XML documents. A global element is a fully-defined element that you can simply append to other elements, while a global type is an element template that you can extend or complete to define a new element. A close analogy may be classes and abstract classes in object oriented programming. The unified instance format vrp-rep_instance_v1.xsd provides a collection of global elements and global types that you can use in different parts of your instance definition. For example, you can use global element `tw` to define availability time windows on links (see §3.5.2) or customer time windows (see **??**). In the remainder of this section we discuss the most important global elements and types included in vrp-rep_instance_v1.xsd.

3.2.1 The `positive_double` and `probability` simple types

In XML jargon a simple type is a type that contains only text (i.e., values) but cannot contain other elements. vrp-rep_instance_v1.xsd defines two simple types: `positive_double` and `probability`. As its name suggest, the `positive_double` simple type defines a positive double value. Elements extending this type can only hold double values that are equal or greater than zero. The `probability` simple type defines the value for a probability, that is a double value $0 \leq v \leq 1$.

3.2.2 The `time_dependent_parameter_type` type

3.2.3 The `dimensions_type` type and the `dimensions` element

The `dimensions_type` type allows you to define elements representing the 2 or 3 dimensional size an object. Element `dimensions` provides a concretization of the `dimensions_type` type. The element has two mandatory (`width` and `height`) and one optional (`depth`) child elements. All these elements extend the `positive_double` simple type introduced in §3.2.1. The `dimensions` element can become handy to

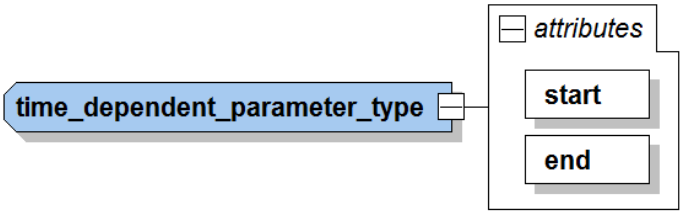

Figure 4: Tree view of the `time_dependent_parameter_type` type

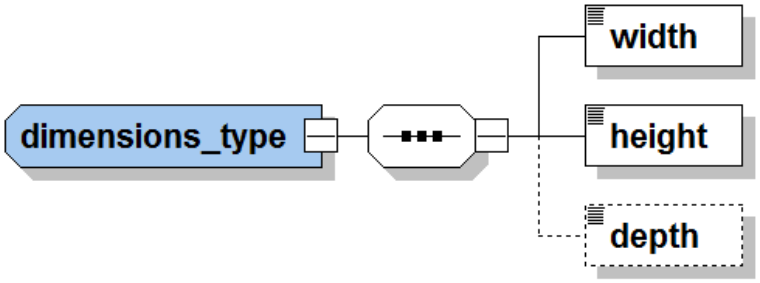

Figure 5: Tree view of the `compartment` element

define features in VRPs with 2 and 3 dimensional loading constraints (e.g., the size of the vehicle and the size of the customer orders). The following listings present examples for the 2D and 3D cases.

Listing 1: Defining 2D dimensions

```
<dimensions>
   <width>200</width>
   <height>250</height>
</dimensions>
```

Listing 2: Defining 3D dimensions

```
<dimensions>
   <width>200</width>
   <height>250</height>
   <depth>500</depth>
</dimensions>
```

3.2.4 The `random_variable` element

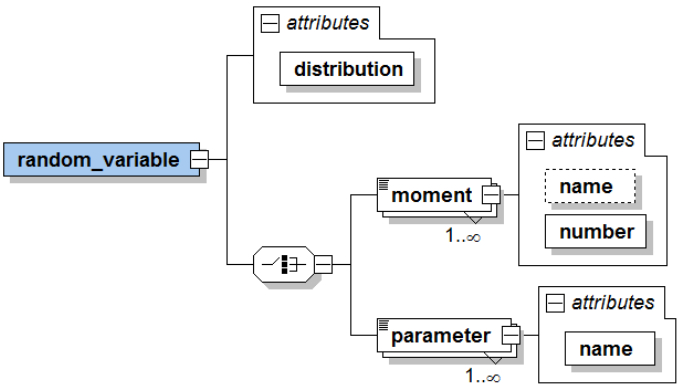

Figure 6: Tree view of the `random_variable` element

The `random_variable` element defines, no surprise, a random variable. This element becomes handy when you need to model VRP features such as stochastic demands, travel times, or service times. The element has one mandatory string attribute, `distribution`, defining the family of distributions to which

the random variable belongs (e.g., normal, Poisson, Erlang). You can define a random variable using one of two types of elements: `moment` or `parameter`. The `moment` element allows you to describe the random variable using its moments (e.g., mean, variance, skewness). A `moment` element is completely defined by an integer attribute `number` (e.g., 1 for the mean, 2 for the variance, 3 for the skewness, 4 for the kurtosis) and a value. In addition you can add the `name` of the moment to the attributes of `moment`. The following example shows how to define a normal distribution using its moments:

Listing 3: Defining a normal distribution with mean equals to 100.56 and variance equals to 5.50

```
<random_variable distribution="normal">
   <moment number="1">100.56</moment>
   <moment number="2">5.50</moment>
</random_variable>
```

The `parameter` element allows you to describe the random variable using the parameters of its underlying distribution. A parameter is completely defined by a string attribute `name`, providing the name of the parameter, and a double value. The following example shows how to define a random variable using the parameter of the Poisson distribution.

Listing 4: Defining a Poisson distribution with lambda equals to 65

```
<random_variable distribution="Poisson">
   <parameter name="lambda">65</parameter>
</random_variable>
```

3.2.5 The `time` element

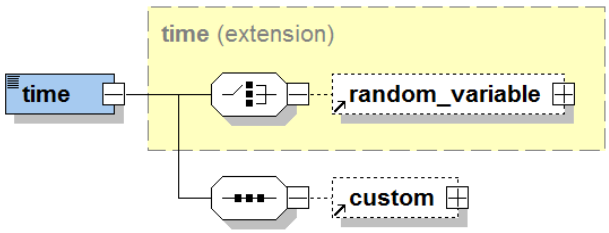

Figure 7: Tree view of the `time` element

The `time` element allows you to model time (e.g., a service time, a travel time to go from A to B). The element is a *mixed content element*, meaning that it may contain either other elements or a single value. Imagine that you want to define the time needed to service a given request (we will talk about requests in §**??**). Assume that this *service time* is equal to 5 time units. The corresponding `time` element looks as follows:

Listing 5: Defining a service time of 5 time units for a given request

```
<time>5</time>
```

Now imagine that the service time is not a deterministic value, but rather a random variable. You can add this feature by appending a `random_variable` element to your `time` element. The following listing shows an example where the service time follows a Poisson distribution with mean 16:

Listing 6: Defining a Poisson distributed service time with mean 16

```
<time>
   <random_variable distribution="Poisson">
      <moment name="mean" number="1">16</moment>
   </random_variable>
</time>
```

Good to know. In XML Schema there is no way to forbid that mixed content element to include both elements and a value. In other words, the following `time` element will be deemed valid by XML validators even if it does not make any sense from a pure VRP perspective:

Listing 7: Example of a valid mixed content element that makes no sense

```
<time>
   5
```

```
   <random_variable distribution="Poisson">
      <moment name="mean" number="1">16</moment>
   </random_variable>
</time>
```

Good practice. Double check your instance generator to avoid having inconsistent mixed content elements in your instance file. Avoid the future users of your files headaches trying to figure out what the data actually means.

The vrp-rep_instance_v1.xsd XML Schema does not define any other features for `time` elements; however, as Figure 7 shows, you can extend the definition of `time` using the `custom` element. We will explain the use of the `custom` elements in Section 3.2.6.

3.2.6 The `custom` element

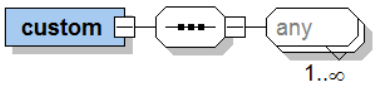

Figure 8: Tree view of the `custom` element

The `custom` element allows you to include in your instance files elements that have not been previously defined in the XML Schema (vrp-rep_instance_v1.xsd). To illustrate how you can use custom elements, let us re-use the example of the service time introduced in §3.2.5. Now imagine that we are in the case of split delivery and that the service takes 20 time units per delivery plus an additional 2 time units per unit of served demand. Obviously, the definition of the `time` element (see §3.2.5) does not allow you to directly model this feature; however, using the `custom` element you can come up with several different solutions. The following listings show a couple of examples:

Listing 8: Modeling non-common features with the `custom` element - fix+variable service time - option 1

```
<time>
   <custom>
      <fix>20</fix>
      <variable>2</variable>
   </custom>
</time>
```

Listing 9: Modeling non-common features with the `custom` element - fix+variable service time - option 2

```
<time>
   <custom>
      <service fix="20" variable="2"/>
   </custom>
</time>
```

Good practice. Try to use elements to describe data, use attributes only to describe meta-data. As it stands, the first example given above is a better choice than the second one. This is a well-spread XML practice.

Good practice. Minimize the use of the `custom` element. Double check that you cannot model a feature using other pre-defined elements before "marrying" a `custom` element. Custom elements are verbose and decrease the readability of your instance files. Custom elements also reduce the compatibility of your instance files with standard data readers and solution checkers developed by the `www.vrp-rep.org` community.

3.2.7 The `tw` element

The `tw` element defines a time window. You can define a `tw` either as a single point or as an interval in the time horizon. To define a single point in time use element `period`. To define an interval in time use elements `start` and `end`. Use boolean attribute `soft` to define if the period, start, and end values of the time windows are soft or hard constraints. The following example shows how to define a time window starting at period 7 and ending at period 10 of the planning horizon. The start time of the time window is a soft constraint while the end time is not.

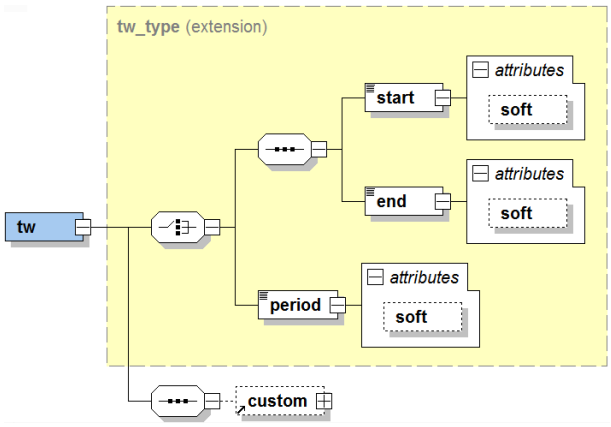

Figure 9: Tree view of the `tw` element

Listing 10: Defining a time window with a flexible start at period 7 and a hard end at period 10

```
<tw>
   <start soft="true">7</start>
   <end soft="false">10</end>
</tw>
```

Good practice. Do not include attribute `is_soft` in your time windows if they are all hard constraints; your instance files will be more readable.

3.2.8 The `cost` element

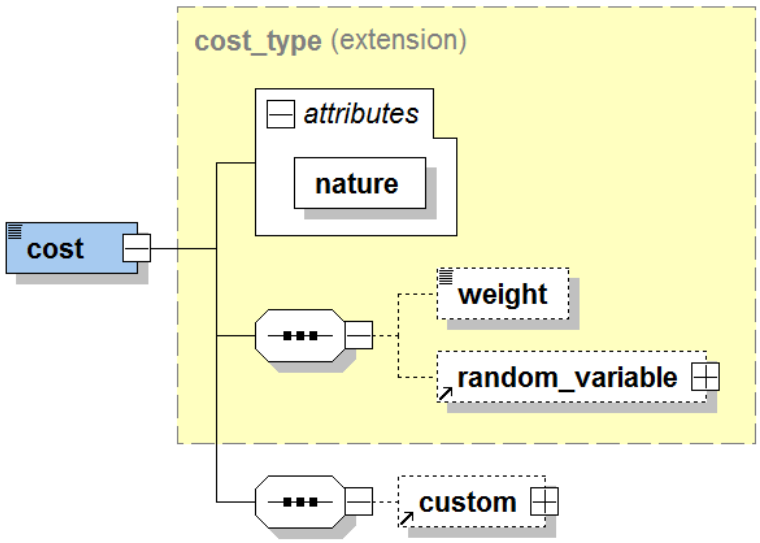

Figure 10: Tree view of the `cost` element

As its name suggests, the `cost` element defines a cost. The element has one mandatory attribute named `nature` that can take only two values "fix" or "variable". If you violate this constraint your instance file will be invalid. Similarly to `time`, `cost` is a mixed content element. The additional elements that you can append to `cost` are `weight` (defining the weight of the cost), `random_variable` (see §3.2.4), and `custom` (see §3.2.6). For illustration, assume that you want to define two costs associated to operating a given type of vehicle (we will talk about vehicle profiles in §3.6). The first cost is a fixed cost of 2,000 for using the vehicle. The second is a variable cost of 1 cost unit per kilometer. The following listings present the two associated `cost` elements.

Listing 11: Modeling a fix cost

```
<cost nature="fix">2000</cost>
```

Listing 12: Modeling a variable cost

```
<cost nature="variable">1</cost>
```

Good to know. An XML file is said to be valid if it contains a document that is conform to a document type (in this case vrp-rep_instance_v1.xsd).

Good to know. When uploading a dataset to `www.vrp-rep.org`if your .zip file contains one invalid instance file the whole dataset will be rejected.

Good practice. Validate offline all your instance files before trying to upload your dataset to `www.vrp-rep.org`. Check the website for links to both open-source and proprietary XML validators.

3.3 The root element

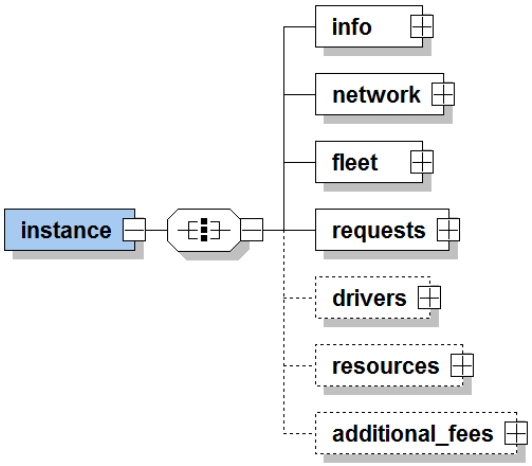

Figure 11: Tree view of the `instance` element

The root element of any instance file is element `instance`. An instance is defined by 4 mandatory elements and 3 optional elements.

- `info`(mandatory): provides information about the instance file.
- `network`(mandatory): defines the underlying graph.
- `fleet`(mandatory): defines the vehicles' profiles.
- `requests`(mandatory): defines the requests to serve.
- `drivers`(optional): defines the drivers' profiles.
- `resources`(optional): defines resources other than vehicles (e.g., tools).
- `additional_fees`(optional): defines costs other than those linked to the vehicles, the drivers, and the requests.

3.4 The `info` element

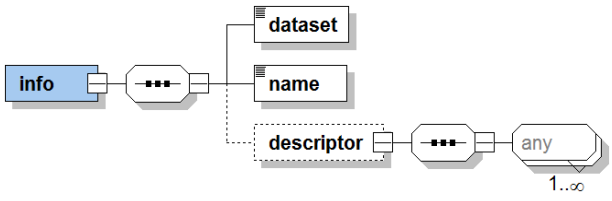

Figure 12: Tree view of the `info` element

The `info` element provides an identification tag for the instance. The element is made up of 2 mandatory elements, `dataset` and `name`, and one optional element `descriptor`. Element `dataset` shall contain the *key* of the dataset. When you upload a dataset to `www.vrp-rep.org`, the website automatically assigns a default key value for the dataset. The website builds the default value using the information of the article in which the dataset was first proposed. The rule to generate the default value is:

- *Author Year* if the article was single authored
- *Author_1 and Author_2 Year* if the article was co-authored by 2 people
- *Author_1 et al. Year* if the article was co-authored by more than 2 people

You can also set your own key value when uploading the dataset.
Element `name` shall contain the name of the instance.

Good practice. Use the default key value for your datasets.

Good to know. The unique identifier of an instance in the `www.vrp-rep.org` database is the name of its XML instance file (without the .xml extension). Therefore, the name you give to an instance file will become the "official" name of the instance in the repository and it cannot be changed.

Good practice. Make the value for element `name` match the name of the instance file (without the .xml extension).

Element `descriptor` allows you to add text or other elements providing a description of your instance file if you feel users need a hand understanding your files.

3.5 The `network` element

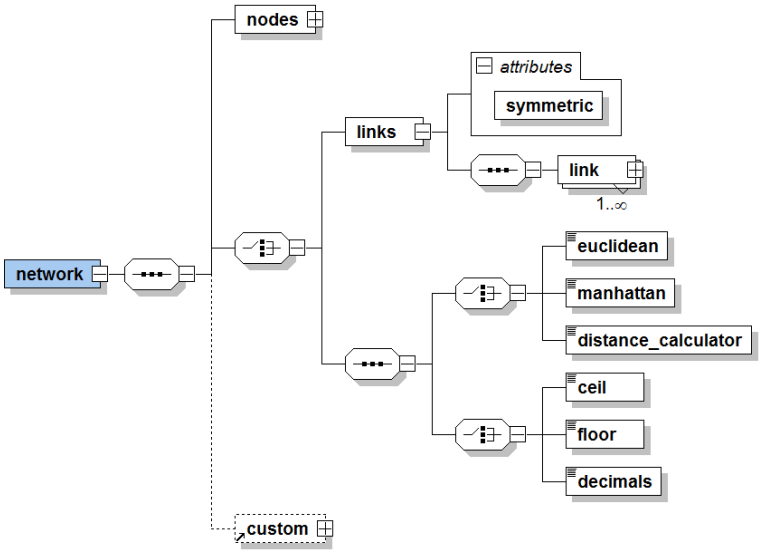

Figure 13: Structure of the `network` element

The `network` element defines the underlying graph. To define the set of nodes, you need at least two `node` elements (grouped under element `nodes`). We will discuss the `node` element in Section 3.5.1. To define the set of links (arcs or edges) you have two alternatives. The first alternative consists on implicity defining the set of links using the nodes. This alternative applies only to complete, symmetric, and undirected graphs and assumes that you provide the location of the nodes (see §3.5.1 for details). The second alternative, which applies to any graph, consists in explicitly defining the set of links using `link` elements. We will discuss this alternative to a larger extend in Section 3.5.2.

To implicitly define the set of links, you need to append to your `network` element two additional elements: one defining how to compute distances and one defining how to round values. There are three elements defining how to compute distances: `euclidean`, `manhattan`, and the more-generic `distance_calculator`. The latter allows you to explain in your own words how distances are to be computed. There are three

pre-defined rounding rules: `ceil` (round up to the next integer), `floor` (round down to the lower integer), and `decimals`. The latter allows you to specify the number of rounding decimals.

Assume that your `nodes` element is well-formed and valid and that it provides the 2D Euclidean coordinates of every node (you will learn how to do that in §3.5.1). The following listings show examples of how to implicitly define the set of links:

Listing 13: Defining a set of links in a network with Euclidean distances and a 2 decimal rounding rule

```
<network>
   <nodes>
         .
         .
         .
   </nodes>
   <euclidean/>
   <decimals>2</decimals>
</network>
```

Listing 14: Defining a set of links in a network with Manhattan distances and a floor rounding rule

```
<network>
   <nodes>
         .
         .
         .
   </nodes>
   <mahattan/>
   <floor/>
</network>
```

Now assume that your `nodes` element is well-formed and valid and that it provides the GPS coordinates of every node. The following listing shows an example of how to implicitly define the set of links:

Listing 15: Defining a set of links using the Haversine formula and a 4 decimal rounding rule

```
<network>
   <nodes>
      .
      .
      .
   </nodes>
   <distance_calculator>
      Haversine formula
   </distance_calculator>
   <decimals>4</decimals>
</network>
```

You can be more specific and explicitly provide the formula:

Listing 16: Defining a set of links using the Haversine formula and a 4 decimal rounding rule

```
<network>
   <nodes>
      .
      .
      .
   </nodes>
   <distance_calculator>
      Haversine formula:
      d=2r\arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)
      +\cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)

      where:
      d is the distance between the two points
      r  is the radius of the earth
      \phi_1, \phi_2 are the latitude of point 1 and latitude of point 2
      \lambda_1, \lambda_2 are the longitude of point 1 and longitude of point 2
   </distance_calculator>
   <decimals>4</decimals>
</network>
```

Good practice. Provide the whole formula only if you think that the eventual user of your files may have trouble finding it by him/herself. Write the formula preferably using LaTeX syntax. On the other hand, if your formula is a fairly-known expression or it can be easily retrieved on the internet (e.g., the

Haversine formula in the example above) just provide the name of the function. This will make your files more readable and lighter.

3.5.1 The `node` element

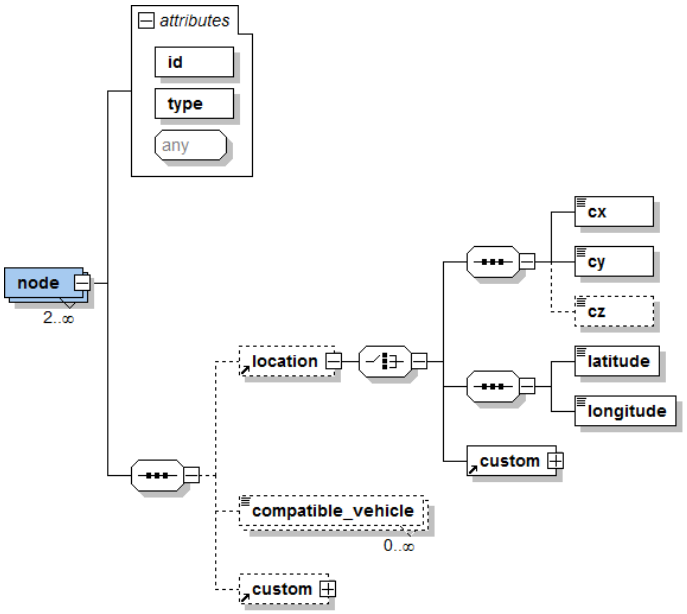

Figure 14: Structure of the `node` element

The `node` element defines a node of the graph. A `node` has two mandatory (integer) attributes `id` and `type`. Attribute `id` provides a unique identifier to a node, meaning that the value you give to this attribute must be unique across all `node` elements. If you violate this constraint, XML validators will deem your instance file invalid. Attribute `type` allows you to define multiple types of nodes (e.g., depots and customers). You can add your own attributes to a `node` using the `any` attribute (see Example 4). You can define the location of the node using either Euclidean coordinates or GPS coordinates. A set of Euclidean coordinates is defined by a pair (`cx`,`cy`) of elements. In addition you may add a `cz` element to represent altitude. A set of GPS coordinates is defined by a `latitude` and a `longitude` element. To complete the dentition of a node you can use two optional elements:

- `compatible_vehicle`: allows you to model vehicle-node compatibility constraints (e.g., site-dependent VRPs). You can define as many compatible vehicle profiles as you want; however, the value you set for a `compatible_vehicle` element must correspond to the `type` of one of the vehicle profiles defined in `fleet` (see §3.6).
- `custom`: allows you to extend the definition of a node by adding your own elements. Details on how to model features using the `custom` element are provided in §3.2.6.

Good to know. To provide information about the distance between two nodes in the network you have two alternatives: i) provide the `location` of each node and let the user compute the distance or ii) provide directly the `length` of each `link` (see §3.5.2).

Good to know. An XML file is said to be valid if it contains a document that is conform to a document type (in this case vrp-rep_instance_v1.xsd).

Good to know. When uploading a dataset to `www.vrp-rep.org`if your .zip file contains one invalid instance file the whole dataset will be rejected.

Good practice. Validate offline all your instance files before trying to upload your dataset to `www.vrp-rep.org`. Check the website for links to both open-source and proprietary XML validators.

Some examples

Example 1. Defining the set of nodes in the classical CVRP. Let $\mathcal{N} = \{0, 1, 2, 3, 4\}$ be the set of nodes in a graph for a given instance of the capacitated vehicle routing problem (CVRP). Node 0 represents the depot and nodes $1 - 4$ represent the customers. Assume that we know the 2D Euclidean coordinates of the five nodes and we want to include them in the instance file. To differentiate the depot from the customers we set attribute `type=0` for the depot node and `type=1` for the customer nodes. The following listing presents an example of the `nodes` element in the corresponding XML instance file.

Listing 17: Example 1 - defining nodes in the CVRP

```
<nodes>
   <node id="0" type="0">
      <cx>5.00</cx>
      <cy>5.00</cy>
   </node>
   <node id="1" type="1">
      <cx>0.00</cx>
      <cy>10.00</cy>
   </node>
   <node id="2" type="1">
      <cx>10.00</cx>
      <cy>0.00</cy>
   </node>
   <node id="3" type="1">
      <cx>10.00</cx>
      <cy>10.00</cy>
   </node>
   <node id="4" type="1">
      <cx>0.00</cx>
      <cy>0.00</cy>
   </node>
</nodes>
```

Example 2. Defining the set of nodes in the Green VRP. Let $\mathcal{N} = \{0\} \cup \mathcal{I} \cup \mathcal{F}$ be the set of nodes in an instance of the Green VRP. Node 0 represents the depot, $\mathcal{I} = \{1, 2, 3\}$ the set of customers, and $\mathcal{F} = \{4, 5\}$ the set of *alternative fueling stations*(AFSs). Assume that we do not know the coordinates of the nodes. To differentiate the three types of nodes we set attribute `type=0` to represent the depot node, `type=1` to represent customer nodes, and `type=2` to represent AFSs. The following listing presents an example of the `nodes` element in the corresponding XML instance file.

Listing 18: Example 2 - defining nodes in the Green VRP

```
<nodes>
   <node id="0" type="0"/>
   <node id="1" type="1"/>
   <node id="2" type="1"/>
   <node id="3" type="1"/>
   <node id="4" type="2"/>
   <node id="5" type="2"/>
</nodes>
```

Example 3. Defining the set of nodes in the site-dependent VRP. Let $\mathcal{N} = \{0, 1, 2, 3, 4\}$ be the set of nodes in a graph for a given instance of the site-dependent vehicle routing problem (SDVRP). Node 0 represents the depot and nodes $1 - 4$ represent the customers. To differentiate the depot from the customers we set attribute `type=0` for the depot node and `type=1` for the customer nodes. Assume that we do not know the coordinates of the customers. Assume also that in the given instance, there exist two types of vehicle profiles (1 and 2). Customer 1 is accessible to both types of vehicle, customers 2 and 4 only to vehicles of type 1, and customer 3 only to vehicles of type 2. The following listing presents an example of the corresponding `nodes` element.

Listing 19: Example 3 - defining nodes in the SDVRP

```
<nodes>
   <node id="0" type="0">
      <compatible_vehicle>1</compatible_vehicle>
      <compatible_vehicle>2</compatible_vehicle>
   </node>
   <node id="1" type="1">
      <compatible_vehicle>1</compatible_vehicle>
      <compatible_vehicle>2</compatible_vehicle>
   </node>
   <node id="2" type="1">
```

```
      <compatible_vehicle>1</compatible_vehicle>
   </node>
   <node id="3" type="1">
      <compatible_vehicle>2</compatible_vehicle>
   </node>
   <node id="4" type="1">
      <compatible_vehicle>1</compatible_vehicle>
   </node>
</nodes>
```

Example 4. Defining the set of nodes in the truck and trailer routing problem (TTRP). Let $\mathcal{N} = \{0\} \cup \mathcal{N}_D \cup \mathcal{N}_C$ be the set of nodes in an instance of the TTRP. Node 0 represents the depot, $\mathcal{N}_D = \{1, 2\}$ the set of *trailer points*, and $\mathcal{N}_C = \{3, 4, 5\}$ the set of customers. A trailer point is a place where the vehicle can detach, park, and leave its trailer while performing a sub-route serving some customers. Assume that we do know the 2D Euclidean coordinates of the nodes. To differentiate the three types of nodes we set attribute `type=0` to represent the depot node, `type=1` to represent trailer points, and `type=2` to represent customers. Assume that customers 3 and 5 can be visited by the truck pulling its trailer, while customer 4 can only be visited by the truck without the trailer. Note that with the standard `node` definition we cannot model truck-trailer accessibility constraints. However, we can extend the `node` definition by adding a boolean attribute `trailer` modeling this feature. The following listing presents an example of the `nodes` element in the corresponding XML instance file.

Listing 20: Example 4 - defining nodes in the TTRP

```
<nodes>
   <node id="0" type="0" trailer="true">
      <cx>5.00</cx>
      <cy>5.00</cy>
   </node>
   <node id="1" type="1" trailer="true">
      <cx>0.00</cx>
      <cy>10.00</cy>
   </node>
   <node id="2" type="1" trailer="true">
      <cx>10.00</cx>
      <cy>0.00</cy>
   </node>
   <node id="3" type="2" trailer="true">
      <cx>10.00</cx>
      <cy>10.00</cy>
   </node>
   <node id="4" type="2" trailer="false">
      <cx>0.00</cx>
      <cy>0.00</cy>
   </node>
   <node id="5" type="2" trailer="true">
      <cx>2.50</cx>
      <cy>2.50</cy>
   </node>
</nodes>
```

Good practice. Minimize the use of non-standard attributes in your `node` definition. Your instance files will be more compatible with standard readers and solution checkers developed by the `www.vrp-rep.org` community.

Good practice. Be consistent in the use of non-standard attributes. This will make the development of data readers and solution checkers a lot easier. For instance, in the example above technically speaking we only need to add `trailer` attribute to customer nodes, since it is clear from the problem definition than the depot and the tailer points are accessible to the truck pulling the trailer. Note, however, that for the sake of consistency we also added the attribute to the depot and the trailer points.

3.5.2 The `link` element

The `link` element models an edge or an arc in the underlying graph. Link elements have two mandatory attributes: `tail_node` and `head_node`. As their names suggest, these attributes denote the two nodes that are connected by the `link`. The values for these two attributes must correspond to the identifiers of nodes previously defined in the `nodes` section (see §3.5.1). To complete the definition of a link you can include the following optional attributes:

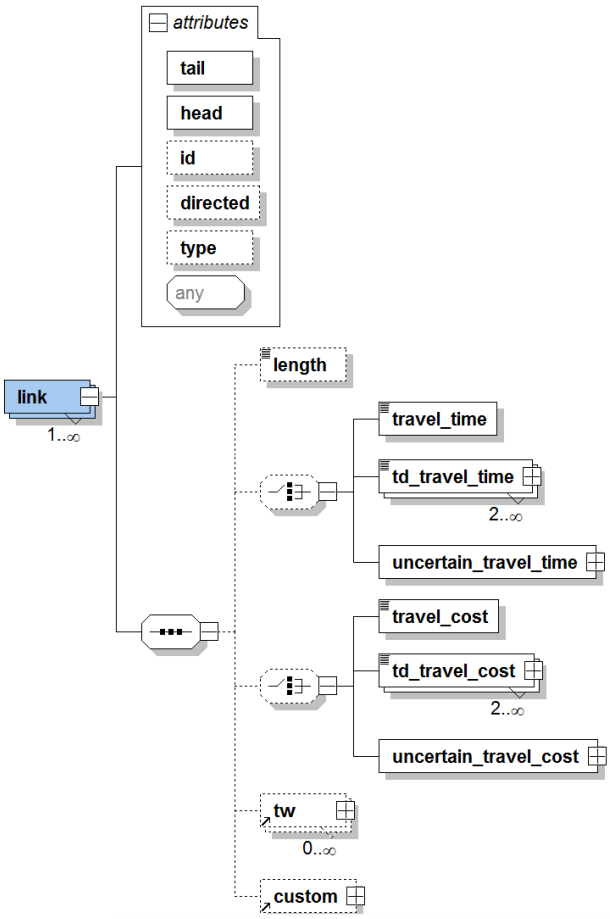

Figure 15: Tree view of the `link` element

- `id`: integer attribute that allows you to define a unique identifier for the link.
- `directed`: boolean attribute that allows you to model directed links (arcs). Set `directed=true` if the link is directed and `directed=false` otherwise.
- `type`: integer attribute that allows you to define a type for the link (e.g., urban-area links vs. highway links).
- `any`: allows you to extend the standard definition of `link`. See Example 4 for an illustration of how to use the `any` attribute.

Good practice. Do not include attribute `directed` in your `link` elements if your graph is undirected. This improves the human readability of your instance files. By default, standard instance readers and solution checkers assume that links are undirected (as it is the case in most VRP variants).

Good practice. Include attribute `directed` in every `link` element if you have both directed and undirected links in your graph. This improves the human readability of your files and makes instance readers and solution checkers easier to code.

Good practice. Include in your file only once the definition of your links if your graph is symmetric (i.e., `links/@symmetric=true`). This considerably reduces the size of your instance files and improves their readability. For illustration, assume that your (symmetric) graph has n nodes with identifiers going from 1 to n. Assuming that your graph is complete, you only need to define the first $\frac{n(n-1)}{2}$ links of the graph. The first arc of the graph is $(1, 2)$ and the $\frac{n(n-1)}{2}$th arc of the graph is $(n-1, n)$.

You can add several optional elements to your `link` definition:

- a `length` element defining the length (in distance units) of the link. This element extends the `positive_double` simple type introduced in §3.2.1.

- a `travel_time` element defining the deterministic and time-independent travel time on the link. This element extends the `positive_double` simple type introduced in §3.2.1.
- two or more `td_travel_time` elements defining the deterministic and time-dependent travel times on the link. These elements extend the `time_dependent_parameter_type` type introduced in §**??**.
- an `uncertain_travel_time` element defining the stochastic travel time on the link. This element extends the `uncertain_parameter` type introduced in §**??**.
- a `travel_cost` element defining the deterministic and time-independent cost of using the link. This element extends the `positive_double` simple type introduced in §3.2.1.
- two or more `td_travel_cost` elements defining the deterministic and time-dependent costs of using the link. These elements extend the `time_dependent_parameter_type` type introduced in §**??**.
- an `uncertain_travel_cost` element defining the stochastic travel cost of using the link. This element extends the `uncertain_parameter` type introduced in §**??**.
- one or more `tw` elements (see §3.2.7) defining the time periods in which the link is available.
- a `custom` element extending the standard definition of `link`.

Some examples

Example 5. Defining the set of edges in the classical CVRP. Let $G = (\mathcal{N}, \mathcal{E})$ be the a complete and undirected graph for a given instance of the capacitated vehicle routing problem (CVRP). Set $\mathcal{N} = \{0, 1, 2, 3, 4\}$ is the set of nodes and set $\mathcal{E}$ is the set of edges. Assume the `nodes` element is well-formed and valid and that it does not provide the coordinates of the nodes. The following listing presents an example of the `links` element in the corresponding XML instance file[1].

Listing 21: Example 5 - defining links in the CVRP

```
<links symmetric="true">
   <link tail="0" head="1">
      <length>7.07</length>
   </link>
   <link tail="0" head="2">
      <length>7.07</length>
   </link>
   <link tail="0" head="3">
      <length>7.07</length>
   </link>
   <link tail="0" head="4">
      <length>7.07</length>
   </link>
   <link tail="1" head="2">
      <length>14.14</length>
   </link>
   <link tail="1" head="3">
      <length>10.00</length>
   </link>
   <link tail="1" head="4">
      <length>10.00</length>
   </link>
   <link tail="2" head="3">
      <length>10.00</length>
   </link>
   <link tail="2" head="4">
      <length>10.00</length>
   </link>
   <link tail="3" head="4">
      <length>14.14</length>
   </link>
</links>
```

Example 6. Defining the set of edges in the asymmetric traveling salesman problem (ATSP). Let $G = (\mathcal{N}, \mathcal{E})$ be the a complete and undirected graph for a given instance of the ATSP. Set $\mathcal{N} = \{0, 1, 2, 3, 4\}$ is the set of nodes and set $\mathcal{E}$ is the set of edges. Assume the `nodes` element is well-formed and valid. The following listing presents an example of the `links` element in the corresponding XML instance file[2].

[1] The lengths of the edges are computed using the coordinates given for the nodes in Example 1

[2] The graph of the example was built using the first five nodes in instance br17 from the TSPLIB

Listing 22: Example 6 - defining links in the ATSP

```
<links symmetric="false">
   <link tail="0" head="1">
      <length>3</length>
   </link>
   <link tail="0" head="2">
      <length>5</length>
   </link>
   <link tail="0" head="3">
      <length>48</length>
   </link>
   <link tail="0" head="4">
      <length>48</length>
   </link>
   <link tail="1" head="0">
      <length>3</length>
   </link>
   <link tail="1" head="2">
      <length>3</length>
   </link>
   <link tail="1" head="3">
      <length>48</length>
   </link>
   <link tail="1" head="4">
      <length>48</length>
   </link>
   <link tail="2" head="0">
      <length>5</length>
   </link>
   <link tail="2" head="1">
      <length>3</length>
   </link>
   <link tail="2" head="3">
      <length>72</length>
   </link>
   <link tail="2" head="4">
      <length>72</length>
   </link>
   <link tail="3" head="0">
      <length>48</length>
   </link>
   <link tail="3" head="1">
      <length>48</length>
   </link>
   <link tail="3" head="2">
      <length>74</length>
   </link>
   <link tail="3" head="4">
      <length>0</length>
   </link>
   <link tail="4" head="0">
      <length>48</length>
   </link>
   <link tail="4" head="1">
      <length>48</length>
   </link>
   <link tail="4" head="2">
      <length>74</length>
   </link>
   <link tail="4" head="3">
      <length>0</length>
   </link>
</links>
```

Good practice. Include a definition for every link if your graph is asymmetric, even if you have some symmetric edges. Although this makes your file more verbose, it will be more readable and more compatible with standard instance readers and solution checkers.

Example 7. Defining the set of links in the time-dependent VRP (TD-VRP). Let $G = (\mathcal{N}, \mathcal{E})$ be the complete and undirected graph for a given instance of the TD-VRP. Set $\mathcal{N} = \{0, 1, 2\}$ is the set of nodes and set $\mathcal{E}$ is the set of edges. Assume the `nodes` element is well-formed and valid. Assume that in normal conditions (off-peak traffic) the travel time on all edges is 20 time units and that all vehicles have a maximum route duration of 60 time units (you will learn how to set this constraint in §3.6). The

following listing presents an example of the `links` element corresponding to a possible instance for the problem.

Listing 23: Example 7 - defining links in the TD-VRP

```
<links symmetric="true">
   <link tail="0" head="1">
      <td_travel_time start="0" end="60">20</td_travel_time>
   </link>
   <link tail="0" head="2">
      <td_travel_time start="0" end="20">25</td_travel_time>
      <td_travel_time start="20" end="60">20</td_travel_time>
   </link>
   <link tail="1" head="2">
      <td_travel_time start="0" end="25">30</td_travel_time>
      <td_travel_time start="25" end="45">20</td_travel_time>
      <td_travel_time start="45" end="60">32</td_travel_time>
   </link>
</links>
```

Note that the structure is flexible to model different situations. For instance, in this example edge (0,1) has actually a time-independent travel time, while edges (0,2) and (1,2) show time dependency. Edge (0,2) exhibits rush-hour traffic in the interval $[0, 20)$ and off-peak traffic in the interval $[19, 60]$. Edge (1,2) exhibits rush-hour traffic in the intervals $[0, 25)$ and $[45, 60]$, and off-peak traffic in the interval $[25, 45)$. Note that the travel time in the two rush-hour traffic intervals for edge (1,2) are not necessarily the same.

Example 8. Defining the set of links in the VRP with stochastic travel times (VRPSTT). Let $G = (\mathcal{N}, \mathcal{E})$ be the complete and undirected graph for a given instance of the VRPSTT. Set $\mathcal{N} = \{0, 1, 2\}$ is the set of nodes and set $\mathcal{E}$ is the set of edges. Assume the `nodes` element is well-formed and valid. Assume that the travel time on edge (0,1) follows an Erlang distribution with shape $k = 1$ and rate $\lambda = 0.1$, the travel time on edge (0,2) follows also an Erlang distribution with $k = 2$ and $\lambda = 0.4$, and the travel time on edge (1,2) follows a Gamma distribution with shape $k = 1$ and scale $\theta = 0.5$. The following listing presents an example of the `links` element in the corresponding XML instance file[3].

Listing 24: Example 8 - defining links in the VRPSTT

```
<links symmetric="true">
   <link tail="0" head="1">
      <uncertain_travel_time>
         <random_variable distribution="Erlang">
            <parameter name="rate">0.1</parameter>
            <parameter name="shape">1</parameter>
         </random_variable>
      </uncertain_travel_time>
   </link>
   <link tail="0" head="2">
      <uncertain_travel_time>
         <random_variable distribution="Gamma">
            <parameter name="scale">0.5</parameter>
            <parameter name="shape">1</parameter>
         </random_variable>
      </uncertain_travel_time>
   </link>
   <link tail="1" head="2">
      <uncertain_travel_time>
         <random_variable distribution="Erlang">
            <parameter name="rate">0.4</parameter>
            <parameter name="shape">2</parameter>
         </random_variable>
      </uncertain_travel_time>
   </link>
</links>
```

Example 9. Defining the set of links in the the with stochastic travel times (VRPSTT). Let $G = (\mathcal{N}, \mathcal{E})$ be the complete and undirected graph for a given instance of the VRPSTT. Set $\mathcal{N} = \{0, 1, 2\}$ is the set of nodes and set $\mathcal{E}$ is the set of edges. Assume the `nodes` element is well-formed and valid. Assume that stochastic travel times are modeled using scenarios rather than probability distributions. Assume that there are 5 possible realizations of the travel time vector and that all 5 scenarios have the same probability. For instance, in scenario one the travel times are 8, 15, and 20 for edges (0,1), (0,2), and (1,2), respectively. The following listing presents an example of the `links` element in the corresponding XML instance.

[3]Example taken from [1]

Listing 25: Example 9 - defining links in the VRPSTT using scenarios

```
<links symmetric="true">
   <link tail="0" head="1">
      <uncertain_travel_time>
         <scenario probability="0.20" id="1">8</scenario>
         <scenario probability="0.20" id="2">8</scenario>
         <scenario probability="0.20" id="3">9</scenario>
         <scenario probability="0.20" id="4">9</scenario>
         <scenario probability="0.20" id="5">9</scenario>
      </uncertain_travel_time>
   </link>
   <link tail="0" head="2">
      <uncertain_travel_time>
         <scenario probability="0.20" id="1">15</scenario>
         <scenario probability="0.20" id="2">15</scenario>
         <scenario probability="0.20" id="3">15</scenario>
         <scenario probability="0.20" id="4">14</scenario>
         <scenario probability="0.20" id="5">14</scenario>
      </uncertain_travel_time>
   </link>
   <link tail="1" head="2">
      <uncertain_travel_time>
         <scenario probability="0.20" id="1">20</scenario>
         <scenario probability="0.20" id="2">21</scenario>
         <scenario probability="0.20" id="3">20</scenario>
         <scenario probability="0.20" id="4">20</scenario>
         <scenario probability="0.20" id="5">21</scenario>
      </uncertain_travel_time>
   </link>
</links>
```

Now assume that the scenarios are not defined for the time travel vector but independently for each edge. The following listing presents an example of the corresponding `links` element.

Listing 26: Example 9 - defining links in the VRPSTT using scenarios

```
<links symmetric="true">
   <link tail="0" head="1">
      <uncertain_travel_time>
         <scenario probability="0.20" id="1">8</scenario>
         <scenario probability="0.20" id="2">9</scenario>
         <scenario probability="0.20" id="3">10</scenario>
         <scenario probability="0.20" id="4">11</scenario>
         <scenario probability="0.20" id="5">12</scenario>
      </uncertain_travel_time>
   </link>
   <link tail="0" head="2">
      <uncertain_travel_time>
         <scenario probability="0.60" id="1">15</scenario>
         <scenario probability="0.40" id="2">17</scenario>
      </uncertain_travel_time>
   </link>
   <link tail="1" head="2">
      <uncertain_travel_time>
         <scenario probability="1.0" id="1">20</scenario>
      </uncertain_travel_time>
   </link>
</links>
```

3.6 The `fleet` element

As its name suggests the `fleet` element defines the fleet of vehicles as a collection of `vehicle_profile` elements. As you can guess, each `vehicle_profile` defines the profile of one type of vehicle.

A `vehicle_profile` has one mandatory and one optional attributes:

- `type` (mandatory): integer attribute defining the type of vehicle profile. This attribute is a key, meaning that your cannot have two `vehicle profile` elements with the same value for this attribute.
- `number` (optional): integer attribute defining the number of available vehicles of the type.

You can also add your own attributes to a `vehicle_profile` using the `any` attribute (see Example 4).

Good practice. Do not add attribute `number` to your `vehicle_profile` element if your fleet is unlimited. Standard data readers and solution checkers assume the fleet is unlimited if the attribute does not appear in the element, you do not need to "cheat" going with `number`=9999999.

3.7 Defining vehicle characteristics

3.7.1 Departing and arrival nodes (mandatory)

For each vehicle profile you must define the node (or set of nodes) from where the vehicles can start their routes, and the node (or set of nodes) where the vehicles can end their routes. To define the departure node(s) you have two possibilities:

- add a `departure_from_any_node` element to your `vehicle_profile` element if vehicles can start their routes from any node of the underlying graph (see §3.5.1)
- add one or more `departure_node` elements listing the `id`s of the nodes from where the vehicles can start their routes

Similarly, to define the arrival node(s) you have two possibilities:

- add an `arrival_at_any_node` element to your `vehicle_profile` element if vehicles can end their routes at any node of the underlying graph (see §3.5.1)
- add one or more `arrival_node` elements listing the `id`s of the nodes where the vehicles can end their routes

Good to know. vrp-rep_instance_v1.xsd includes a constraint to guarantee that the `id`s of the nodes you include in your `departure_node` and `arrival_node` elements are actually the `id`s of nodes defined in the nodes section. If you violate this constraint, your instance files will be deemed invalid by `www.vrp-rep.org`.

Some examples

The following example shows the fleet element for a simple traveling salesman problem (TSP).

Listing 27: Defining departure and arrival nodes in the TSP

```
<fleet>
   <vehicle_profile type="1" number="1">
      <departure_from_any_node/>
      <arrival_at_any_node/>
   </vehicle_profile>
</fleet>
```

Now imagine that you have an open VRP (OVRP) with one type of uncapacitated vehicles parked at node 0 (the depot). The following example shows the corresponding `fleet` element.

Listing 28: Defining departure and arrival nodes in the OVRP

```
<fleet>
   <vehicle_profile type="1">
      <departure_node>0</departure_node>
      <arrival_at_any_node/>
   </vehicle_profile>
</fleet>
```

Now assume that you have a problem variant in which you have two types of vehicles (1 and 2) and two depots (nodes 0 and 1). Vehicles of type 1 must start their routes at node 0 and can end their routes at any of the depots. On the other hand, vehicles of type 2 must start and end their routes at node 1. The following listing shows the corresponding `fleet` element.

Listing 29: Defining departure and arrival nodes in the OVRP

```
<fleet>
   <vehicle_profile type="1">
      <departure_node>0</departure_node>
      <arrival_node>0</arrival_node>
      <arrival_node>1</arrival_node>
   </vehicle_profile>
   <vehicle_profile type="2">
      <departure_node>1</departure_node>
      <arrival_node>1</arrival_node>
   </vehicle_profile>
</fleet>
```

3.7.2 Simple characteristics

Most VRP variants include simple vehicle characteristics such as a global vehicle capacity and a maximum travel time. You can add these simple characteristics as follows:

- Vehicle capacity: you can define a global vehicle capacity using either a `capacity` element or a pair (`max_weight`, `max_volume`) defining the maximum loads in terms of weight units and volume units. All these elements extend the `positive_double` simple type (see §3.2.1)
- Maximum travel time: you can define the maximum travel time using a `max_travel_time` element. This element extends the `positive_double` simple type (see §3.2.1)
- Maximum travel distance: you can define the maximum travel distance using a `max_travel_distance` element. This element extends the `positive_double` simple type (see §3.2.1)
- Vehicle dimensions: you can add vehicle dimensions by appending to your `vehicle_profile` element a `dimensions` element (see §3.2.3)

Some examples

Example 10. Defining vehicle profiles in the classical Distance-constrained CVRP (DCVRP). In the DCVRP the fleet is homogenous and unlimited. As it stands, there is just one vehicle profile to be defined. Vehicles have two constraints: a maximum capacity and a maximum travel distance. The following listing shows an example of how the fleet is defined in a DCVRP[4].

Listing 30: Example 10 - defining the fleet in the DCVRP

```
<fleet>
   <vehicle_profile type="1">
      <departure_node>0</departure_node>
      <arrival_node>0</arrival_node>
      <capacity>6000</capacity>
      <max_travel_distance>210</max_travel_distance>
   </vehicle_profile>
</fleet>
```

Now imagine that you want to set a limitation on the number of vehicles. You can add this constraint by simply adding a `number` attribute to the vehicle profile as shown in the following listing.

Listing 31: Example 10 - defining the fleet in the DCVRP

```
<fleet>
   <vehicle_profile type="1" number="4">
      <capacity>6000</capacity>
      <max_travel_distance>210</max_travel_distance>
   </vehicle_profile>
</fleet>
```

Another simple characteristic that you can add to your `vehicle_profile` is the `speed factor` of the vehicle. This element becomes handy when different types of vehicles have different speed profiles. The following listing shows an example.

Listing 32: Defining vehicle speed profiles

```
<fleet>
   <vehicle_profile type="1" number="4">
      <capacity>6000</capacity>
      <max_travel_time>210</max_travel_time>
      <speed_factor>1.0</speed_factor>
   </vehicle_profile>
   <vehicle_profile type="2" number="6">
      <capacity>3000</capacity>
      <max_travel_time>210</max_travel_time>
      <speed_factor>1.5</speed_factor>
   </vehicle_profile>
</fleet>
```

In the example above, vehicles of type 1 have a speed factor of 1 while vehicles of type 2 have a speed factor of 1.5. In practice these speed factors mean that if the nominal travel time on an edge (i, j) is $t_{i,j}$ (see §3.5.2) then vehicles of type 1 take exactly $t_{i,j}$ time units to travel the edge, while vehicles of type 2 take only $\frac{t_{i,j}}{1.5}$ time units.

[4]Example taken from instance D022-04g in the VRPLIB

3.7.3 Compartments

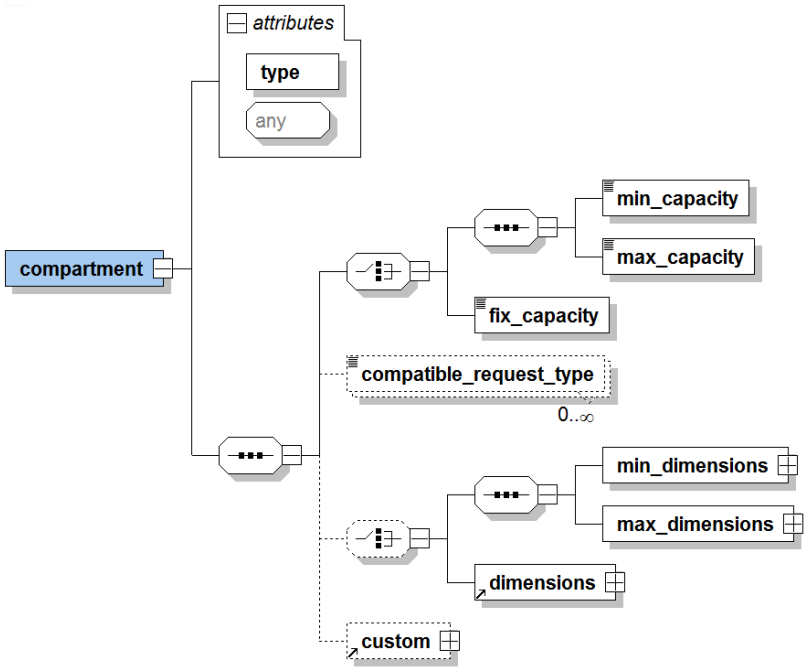

Figure 16: Tree view of the `compartment` element

The `compartment` element allows you to define a vehicle compartment. The element has one mandatory integer attribute named `type` defining, no surprise, the type of compartment. You can also add your own attributes using the `any` attribute (see Example 4).

Capacity: compartments can have either a fixed or a flexible capacity. You can add a fixed capacity to your `compartment` element by appending to it a `fix_capacity` element. This element extends the `positive_double` simple type (see §3.2.1). If you want to define a flexible capacity, you need to append two elements to `compartment`: `min_capacity` and `max_capacity`. As their name suggest, these elements define the minimum and maximum capacity of the compartment.

Example 11. Defining vehicle profiles in the multicompartment vehicle routing problem (MCVRP). In the MCVRP a fleet of vehicles carries products that because of incompatibility constraints must travel in independent compartments. In the simplest version of the problem the fleet is unlimited and homogeneous and all routes start and end at the depot (node 0). The following listing shows the definition of the fleet of vehicles in such case[5].

Listing 33: Example 11 - defining the fleet in the MCVRP

```
<fleet>
   <vehicle_profile type="1" number="4">
      <departure_node>0</departure_node>
      <arrival_node>0</arrival_node>
      <max_travel_distance>466</max_travel_distance>
      <compartment type="1">
         <fix_capacity>959</fix_capacity>
      </compartment>
      <compartment type="2">
         <fix_capacity>946</fix_capacity>
      </compartment>
      <compartment type="3">
         <fix_capacity>975</fix_capacity>
      </compartment>
   </vehicle_profile>
</fleet>
```

In some versions of the problem the compartments have flexible capacity, meaning that the capacity of each compartment is a decision variable. In general flexible compartments have both a minimum

[5]Example taken from instance instance 001.C50.P3.LS.T10 in [3]

and a maximum size and the sum of the capacities assigned to the compartments cannot exceed the global capacity of the vehicle. The following listing presents an example defining a vehicle profile with a maximum capacity of 1,000 units and two flexible capacity compartments; each compartment with an adjustable capacity ranging from 200 to 600 units.

Listing 34: Example 11 - defining the fleet in the MCVRP with flexible compartments

```
<fleet>
   <vehicle_profile type="1" number="4">
      <departure_node>0</departure_node>
      <arrival_node>0</arrival_node>
      <capacity>1000</capacity>
      <max_travel_distance>600</max_travel_distance>
      <compartment type="1">
         <min_capacity>200</min_capacity>
         <max_capacity>600</max_capacity>
      </compartment>
      <compartment type="2">
         <min_capacity>200</min_capacity>
         <max_capacity>600</max_capacity>
      </compartment>
   </vehicle_profile>
</fleet>
```

Compatibility constraints: You can define product-compartment compatibility constraints using `compatible_request_type` elements. You first need to add a `type` attribute to each `request` elements in the `requests` section (you will learn how to do that in §**??**). Then you can append to your `compartment` element one `compatible_request_type` element for each type of compatible request.

Example 12. Defining incompatibility constraints in the MCVRP. In this example vehicles serve the demand for three types of products, namely, 1, 2, and 3. Each vehicle has three compartments, each belonging to one type. Compartments of type 1 and 3 can transport any kind of products. On the other hand, compartments of type 2 can only transports products of type 1 and 3. The following listing presents the definition of the corresponding vehicle profile[6].

Listing 35: Example 12 - defining incompatibility constraints in the MCVRP

```
<fleet>
   <vehicle_profile type="1" number="4">
      <departure_node>0</departure_node>
      <arrival_node>0</arrival_node>
      <compartment type="1">
         <fix_capacity>300</fix_capacity>
         <compatible_request_type>1</compatible_request_type>
         <compatible_request_type>2</compatible_request_type>
         <compatible_request_type>3</compatible_request_type>
      </compartment>
      <compartment type="2">
         <fix_capacity>300</fix_capacity>
         <compatible_request_type>1</compatible_request_type>
         <compatible_request_type>3</compatible_request_type>
      </compartment>
      <compartment type="3">
         <fix_capacity>300</fix_capacity>
         <compatible_request_type>1</compatible_request_type>
         <compatible_request_type>2</compatible_request_type>
         <compatible_request_type>3</compatible_request_type>
      </compartment>
   </vehicle_profile>
</fleet>
```

Dimensions: You can define the dimensions of the compartment by appending to your `compartment` element either: i) a `dimensions` element (see §3.2.3) defining the fixed dimensions of the compartment, or ii) a pair of elements (`min_dimensions`, `max_dimensions`) defining, no surprise, the minimum and maximum dimensions of the compartment. Both `min_dimensions` and `max_dimensions` extends the `dimensions_type` type introduced in §3.2.3.

Other characteristics: You can extend the standard definition of a compartment using the custom element (see §3.2.6)

[6]This example is taken from Figure 2.3 in [2]

3.7.4 Resources

Resources model physical objects or skills
In VRP variants like the technician routing problem (see for instance [4]

References

[1] A. Gómez, R. Mario, R. Akhavan-Tabatabaei, A. Medaglia, and J. E. Mendoza. On modeling stochastic travel and service times in vehicle routing. Working paper., 2014.

[2] R. Lahyani. *Unified matheuristic for solving rich vehicle routing problems.* PhD thesis, École Centrale de Lille, 2014.

[3] J. E. Mendoza, B. Castanier, C. Guéret, A. L. Medaglia, and N. Velasco. A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. *Computers & Operations Research*, 37(11):1886–1898, 2010.

[4] Victor Pillac, Christelle Guret, and Andrs L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535, 2013.