The API of KWebS

KWebS provides a simple to use API, it consists of only three operations. While the main operation `graphLayout` provides the core functionality, you can use the operations `getServiceData` and `getPreviewImage` to retrieve the meta data that describes the layout a server provides. In the following you will see how you can use these operations to integrate automatic layout into your application. Although the public interface is declared by the contract of the SOAP web service KWebS uses, we will use the java notation because it fosters understandability.

The Operation `graphLayout`

The central operation of KWebS is `graphLayout`, which is shown in the listing below. It calculates the layout of a graph that you deliver in serial notation as the `serializedGraph` parameter. This parameter is declaring a general serial notation. The server has to know the underlying meta model and form of serialization, therefore, you have to define the *format* you used to submit your graph with the `informat` parameter. In general, when the server has calculated the layout, it delivers the graph for which the layout was calculated in the same format that you chose to submit your source graph. Alternatively, you can define a different output format with the parameter `outformat` and the server performs the necessary translation.

There is a wiki page explaining the currently supported graph formats.

```
/**
 * Calculating the layout of a graph with KWebS
 *
 * @param serializedGraph the serial notation of the source graph
 * @param informat the format of the source graph
 * @param outformat the optional format of the result
 * @param options the optional layout options
 *
 * @return the serial notation of the graph with calculated layout either
 *         in the format of the source graph or a format chosen by the user
 */
String graphLayout(
    String serializedGraph,
    String informat,
    String outformat,
    List<GraphLayoutOption> options
);
```

The optional parameter `options` gives you the possibility to declare a set of layout options. While the algorithm to be used for calculating the layout of the graph is the most important one, you can choose among a variety of other options, e.g. the orientation of edges or the spacing between nodes. Each option is identified by a string based identifier and a corresponding string based value. The following listing shows the according `GraphLayoutOption` dat a type of the API provided by KWebS.

```
public class GraphLayoutOption {
    public String id;
    public String value;
}
```

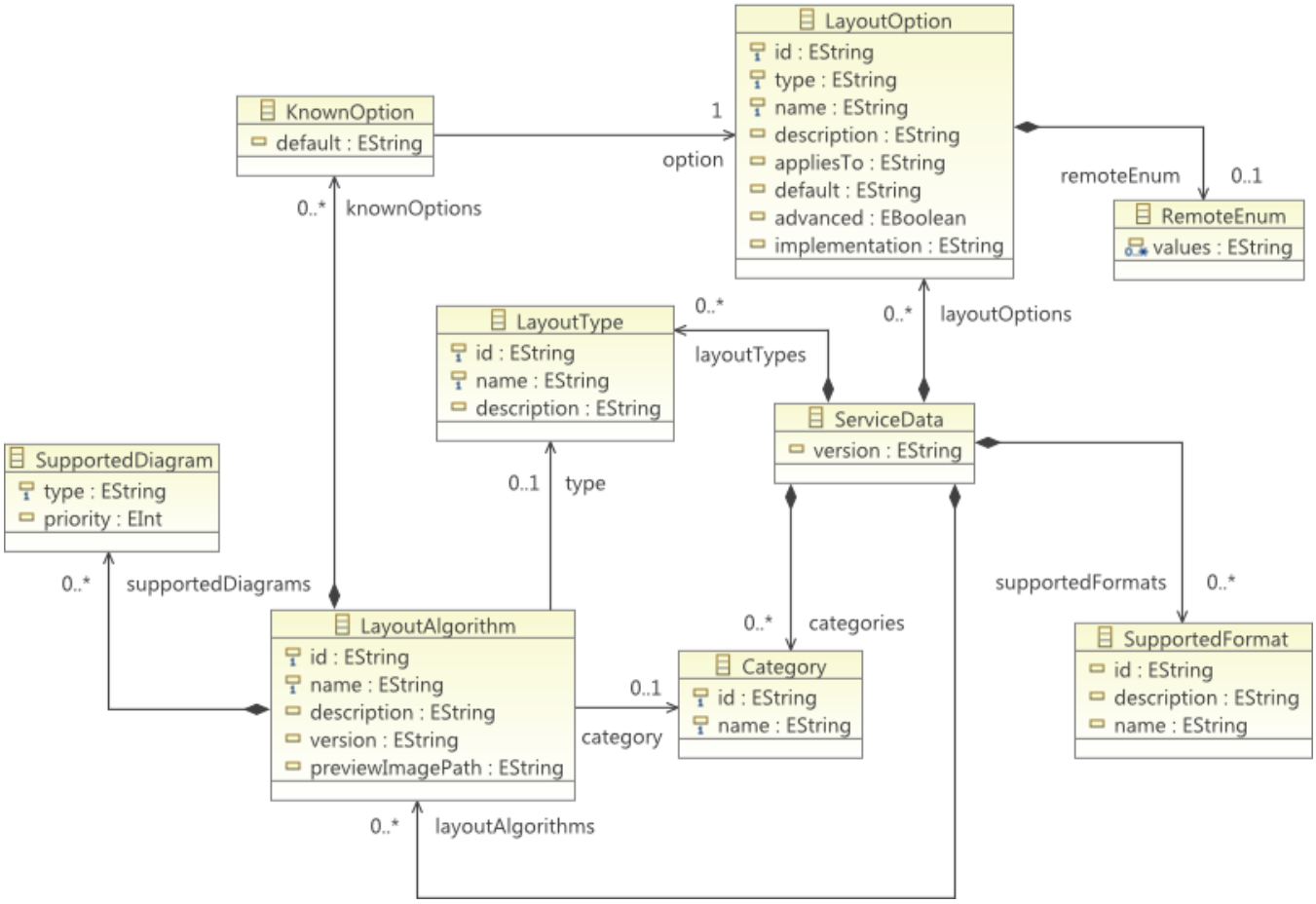
The layout options a server supports are part of the services meta data that are discussed below. Each layout option is connected to a specific type of graph element. While one option only refers to nodes, another option is only applicable to edges. When you define layout options via the `options` paramet er, the server applies each declared option to every compatible graph element. If you want to specify layout options to a specific graph element, you have to do this in the source graph. The server will not overwrite options declared in the source graph if the same option is part of the options list.

The operation `getServiceData`

KWebS leverages the layout of the KIELER project, which is being developed actively, therefore the provided layout may change over time due to additions, removements, or updates on layout algorithms, options, and supported formats. Furthermore, due to its open source and Eclipse nature, providers may decide to add own implementations of the before mentioned components or to support only a subset of these. Therefore, each service instance provides meta data about the layout it supports. You can retrieve the meta data by invoking the operation `getServiceData`, which is shown in the following listing.

```
String getServiceData();
```

When you use this operation, you receive a XML notation of the meta data. It is based on an *Ecore* meta model named *ServiceData*, which is shown by the next figure, and reuses the structure of the extension point `layoutProviders` of KIML.



At startup time, the server initializes a model instance of ServiceData from the extensions the layout related plugins from KIELER make to `layoutProviders`. Additionaly, it adds service related information, e.g. the supported formats, to the model instance. When you request the meta data of a service, you receive the serial notation of the ServiceData model instance in XMI. By providing a structured model for its meta data, KWebS eases the integration of the service-based layout in software systems on the programmatic level.

The structure of the ServiceData meta model is quite simple. The core element is `ServiceData`. Its attribute `version` declares the runtime version of the server. It contains the following children:

- `LayoutAlgorithm` declares a layout algorithm supported by the server. Besides the attributes `name` and `description`, which provide the name and a textual description of the layout the algorithm calculates, the attribute **id** declares the identifier that can be used in combination with the layout option identified by `de.cau.cs.kieler.algorithm` to specify which algorithm the server shall use when you invoke the layout by using its `graphLayout` operation.
- `LayoutType` declares the general type of layout an algorithm computes, e.g. *circular* or *orthogonal* layout. Again, the attributes `name` and `description` provide the name and additional textual information.
- `LayoutOption` declares a layout option. The attribute `type` defines the type of values you may assign to this option, e.g. boolean, string or enumeration values. Additionaly, the attribute `default` may carry a default value which generally provides good layout results. As said before, a layout option is related to specific types of graph elements, specified by the attribute `appliesTo`. It may be empty, meaning, an option can be applied to any element, or contain a comma separated list of compatible element types. The attribute `id` declares the identifier of the layout option. You can use it to specify a layout option directly in the source graph by annotating a specific graph element. The way you realize the annotation is dependent on the model you use. For example, if you use a KGraph model, you could do it like the following listing shows:

```
// Retrieve the graph model somehow
KNode graph = ...

// Get the layout related information from the root node
KShapeLayout shapeLayout = graph.getData(KShapeLayout.class);

// Create a property which is used to define the layout algorithm and
// use KLay layered as the default algorithm
IProperty<String> algorithm = new Property<String>(
    "de.cau.cs.kieler.algorithm",
    "de.cau.cs.kieler.klay.layered"
);

// Annotate the root node of the graph model with the layout option
// represented by the property and use the default algorithm KLay
// layered
shapeLayout.setProperty(algorithm, algorithm.getDefault());

// Alternatively, you can use a different algorithm like the graphviz DOT
shapeLayout.setProperty(algorithm, "de.cau.cs.kieler.graphviz.dot");
```

You could alternatively use the `options` parameter of the `graphLayout` operation to declare a specific layout algorithm, as you can see in the next listing. The `layoutServicePort` instance the example uses resembles the *proxy* necessary to interact with the layout server. You will see how you can create such a proxy in How to use the service based layout in your project.

```
// Retrieve the graph model somehow
KNode graph = ...

// Get the models XMI notation
String source = modelToXmi(graph);

// Create the options list
List<GraphLayoutOption> options = new ArrayList<GraphLayoutOption>();

// Declare the option defining the algorithm to be used
GraphLayoutOption option = new GraphLayoutOption();

option.setId("de.cau.cs.kieler.algorithm");
option.setValue("de.cau.cs.kieler.klay.layered");

// Add the option declaring the layout algorithm to the list
options.add(option);

// Invoke the service
String result = layoutServicePort.graphLayout(
    source,                    // the source model
    "de.cau.cs.kieler.kgraph", // we use the KGraph format
    null,                      // we want the service to return the result in KGraph format
    options                    // we use the options list to specify the algorithm to be used
);

// Get the layouted model
KNode layout = xmiToModel(result);
```

When an option represents an enumeration, it associates a `RemoteEnum` element. It defines the possible assignments you can make to the option.

- `Category` declares a family of algorithms, e.g. KIELER or graphviz.
- `SupportedFormat` declares a format that the service supports, e.g. KGraph, GraphML or DOT. The attribute `id` declares the identifier of the format, which you can use for the `informat` and the `outformat` parameter of the `graphLayout` operation to specify the format of the input graph and the format in which the service shall deliver the layouted graph.

In general, a layout algorithm supports only a subset of the available layout options. The ServiceData meta model considers this fact by associating a `LayoutAlgorithm` to its supported `LayoutOptions` with the `KnownOption` element, which may hold an alternative default value better suited for the needs of the algorithm. Similarly, the association to a `LayoutType` and `Category` element specifies the type of layout and the family it belongs to.

The operation `getPreviewImage`

A user of graphical modeling is not necessarily familiar with the concepts behind the layout of graphs. To help him choosing an algorithm suited to his needs, a service provides additional information that gives a quick overview of the way an algorithm works. Besides a textual description, which is included in the services meta data, he provides *preview images*, which give a visual impression. Due to the binary nature of these images and their size, they are not part of the meta data and have to be downloaded separately. For this purpose, the `LayoutAlgorithm` element of the meta data contains the attribute `previewImage`. It declares an identifier that you can use in combination with the operation `getPreviewImage`, which the following listing shows, to acquire the binary content of the image:

```
byte[] getPreviewImage(String previewImage);
```

Most platforms provide support for creating and displaying images out of their binary representation. Therefore, the `getPreviewImage` operation gives an easy way to integrate a visual representation of the layout an algorithm calculates in the GUI of a software system. This leverages the comprehension of its users and helps increasing their efficiency.