The Ubiquitous Tracking Query Language (UTQL) Version 1.0

Daniel Pustka, Manuel Huber, Peter Keitler, Florian Echtler

October 25, 2007

Abstract

Centrally coordinated *Ubiquitous Tracking* (Ubitrack) setups consist of a Ubitrack server and many clients. The server maintains a central *Spatial Relationship Graph* (SRG) that describes coordinate frames, trackers and trackable objects. Using *Spatial Relationship Patterns*, the server derives new spatial relationships from the SRG and generates data flow descriptions that are executed by the clients to satisfy their own or other client's queries.

In this context, a language is needed for the communication between the clients and the server to exchange descriptions of trackers, tracked objects, tracked or otherwise known spatial relationships, client capabilities and data flow networks. This document describes the *Ubiquitous Tracking Query Language* (UTQL), an XML-based description and query language for Ubitrack system, that fulfils these requirements.

Contents

1 The Ubiquitous Tracking Query Language

The purpose of the Ubitrack Query Language (UTQL) is the communication between a Ubitrack server and its clients. Clients can be sensors providing information to the tracking system or applications, which typically request information. Mixed clients that both provide and consume tracking data also are possible, e.g. virtual characters that react to their environment.

The design builds on the assumption that the Ubitrack server maintains a database storing the Spatial Relationship Graph (SRG). Nodes in the SRG will correspond to entities with geometric meaning, such as physical objects (sensors, locatables) and to virtual enties, such as reference coordinate systems. Nodes will have a number of standard attributes such as id, coordinate system, type, locale etc. Edges in the SRG express direct(-ly measured) or indirect geometric relationships between nodes, and also have QOS attributes (frame rate, accuracy).

1.1 Requirements

UTQL needs to support the following operations:

SRG Sub Graph Registration Clients need to be able to update the global SRG with the nodes and edges they provide.

Query Registration Clients can query for arbitrary sub graphs of the global spatial relationship graph. This includes queries for single nodes, edges or more complex parts of the SRG.

SR Pattern Registration The application of Spatial Relationship Patterns results in components being instantiated in data flow network residing at the client. In order to support clients with different capabilities and Ubitrack versions, the server needs to know about the patterns (data flow components) each client supports.

Removal of Sub-graphs, Queries and Patterns Some method has to be provided to remove sub graphs, queries and patterns from the Ubitrack server database when a client or a tracked object leaves the Ubitrack system.

Dataflow Construction The protocol should be able to express instantiation of data flow components and their connections.

2 Basic conversation structure

The protocol between the client and the server is the same in both directions. A single message looks like this:

```
<UTQLRequest>

<Pattern name="...">
  <Input> ... </Input>
  <Output> ... </Output>
  <Constraints> ... </Constraints>
  <DataflowConfiguration> ... </DataflowConfiguration>
</Pattern>

<Pattern name="...">
  ...
</Pattern>

...

</UTQLRequest>
```

The basic building block of the protocol is a `Pattern` tag which describes an atomic operation in the server's Spatial Relationship Graph. In one burst of communication, multiple `Pattern` tags are enclosed in a `<UTQLRequest>` element. In the Ubitrack system, by registering queries, clients express their interest in particular parts of the global SRG. We can, therefore, also specify server responses in the same language, i.e. as updates of the client's local SRG.

In the client-to-server communication, we can distinguish between three different types of patterns which correspond to the first three requirements:

SRG Updates describe additions, removals or replacements of parts of the basic SRG that is available independently of application queries. It consists mainly of the tracker edges and nodes. Patterns that describe SRG updates only contain `Output` and optionally `DataflowConfiguration` tags.

SR Patterns describe operations that can be performed on the SRG to infer relationships. They contain `Input` and `Output` tags.

Queries only contain `Input` tags and represent the ends of chains of inferred relationships.

`Pattern` structures sent to a client or server are registered in the receiver's internal database until they are removed either explicitly or implicitly by a dropped network connection. If a received `Pattern` conflicts with a previously registered pattern that has the same name or id, the existing pattern is replaced with the new one sent over the connection. This way it is possible to remove queries, patterns, dataflow and SRG parts by replacing them with an empty `<Pattern/>`. By examining the content of the `Input` and `Output` sections, four different types of patterns can be distinguished, which are summarized in the following table:

	`<Input>` empty	`<Input>` non-empty
`<Output>` empty	Remove registered pattern	Query
`<Output>` non-empty	Register sub-graph	SR pattern

3 Description of `Pattern` tags

A `Pattern` structure can contain `Input`, `Output`, `Constraints` and `DataflowConfiguration` tags which are described in more detail in the following sections.

Input Nodes and edges that must be present in the SRG before the sub graph can be instantiated. The nodes and edges contain `Predicate` tags that specify if this condition is reached.

Output Nodes and edges that can be added to the SRG if the inputs are available. The attributes of output nodes/edges may refer to attributes of input nodes/edges using `AttributeExpression` descriptions.

Constraints Additional information for the pattern detection algorithm, e.g. about correspondences for advanced patterns.

DataflowConfiguration This section is not parsed by the Ubitrack server, but passed back to the client when a component needs to be instantiated in the data flow network. Clients should specify everything they need to instantiate the component. This part is client-specific and will contain different information for OpenTracker and UbitrackLib clients.

3.1 Description of `Input` tags

`Input` tags describe the part of the SRG, i.e. the nodes and edges, which must be present before the operation described by the `Pattern` tag can be

performed, i.e. the output edges and nodes can be inserted and/or the data flow components instantiated.

As the `Input` section actually contains a sub-graph, it contains `<Node>` and `<Edge>` tags to describe a graph structure. The following XML snippet gives an example of a query for a single 6D edge going from a node of id "Barn" to a node of id "Sheep":

```
<Input>
  <Node name="Barn">
    <Predicate>id=='Barn'</Predicate>
  </Node>

  <Node name="Sheep">
    <Predicate>id=='Sheep'</Predicate>
  </Node>

  <Edge name="SheepPosition" source="Barn" destination="Sheep">
    <Predicate>type=='6D'</Predicate>
  </Edge>
</Input>
```

Both nodes and edges have `name` attributes which are used to refer to them within the `Pattern` structure. For example, edges use the names of the nodes in their `source` and `destination` attributes. See 4.1 for more details on the scope of names. To match against attributes of nodes and/or edges in the Spatial Relationship Graph, predicates as specified in section 4.2 are used. If no predicates were specified, nodes/edges would match every node/edge in the SRG.

3.2 Description of `Output` tags

The `Output` section describes the nodes and edges that can be added to a Spatial Relationship Graph provided the inputs are present. However, instead of specifying predicates to be matched, the `Node` and `Edge` tags contain attributes that other patterns can match their inputs against. The following gives an example of a single edge that would match the inputs specified in the previous section:

```
<Output>
  <Node name="MySheep" id="Sheep">
    <Attribute name="color" value="black"/>
    <Attribute name="objectType" value="sheep"/>
  </Node>
```

```
  <Node name="Barn" id="Barn"/>

  <Edge name="BlackSheepPose" source="Barn" destination="MySheep">
    <Attribute name="type" value="6D"/>
  </Edge>
</Output>
```

`Output` sections can contain both nodes and edges, but edges can also reference nodes specified in the `Input` part of the same `Pattern`.

Nodes in the `Output` part have a name attribute, just as in the input, but they can also specify the special "`id`" attribute. All nodes that have the same `id` will be considered the same node, no matter in which `Pattern` or by which client they were specified. This way, multiple patterns can be connected to form the complete SRG in the server's database. When nodes have the same `id`, the different attribute sets of the node are merged.

In addition to specifying attributes statically, they can be specified using the `AttributeExpression` tag which allows attribute values to be constructed by referring to attributes of nodes or edges in the `Input` section:

```
<Pattern name="MySheep">
  <Input>
    <Node name="MySheep">
      <Predicate>objectType==’sheep’</Predicate>
    </Node>
  </Input>

  <Output>
    <Node name="Hat">
      <Attribute name="objectType" value="hat"/>
      <AttributeExpression name="color">
        MySheep.color
      </AttributeExpression>
    </Node>
  </Output>
</Pattern>
```

The expression above would add for every node where `objectType` has a value of `'sheep'` another node of `objectType='hat'` which has the same colour as the sheep.

3.3 Description of `Constraints` tags

The `Constraints` section is used to specify additional conditions that must be fulfilled before the operation of the `Pattern` can be performed. The

`Correspondence` constraint is used to specify corresponding measurements by the edges in the `Input` section:

```
<Pattern name="ConstraintExample">
  <Input>
    <Node name="A"/>
    <Node name="B"/>
    <Node name="C"/>
    <Edge name="A2B" source="A" destination="B"/>
    <Edge name="A2C" source="A" destination="C"/>
  </Input>

  <Constraints>
    <Correspondence name="ManyObjects" minMultiplicity="3">
      <Node node-ref="B"/>
      <Edge edge-ref="A2B"/>
      <Edge edge-ref="A2C"/>
    </Correspondence>
  </Constraints>
</Pattern>
```

In addition to `minMultiplicity`, UTQL also supports `maxMultiplicity` and `stepSize` attributes to further constrain the required number of correspondences. More information on the concept of spatial relationship patterns and correspondences in general can be found in [**?**].

The `OnlyBestEdgeMatch` constraint can be used to limit the number of returned pattern instances for each matching set of nodes to the instance with the best-quality edges. This is only useful for queries. In the future, a method to specify what "best" means in this case, will be added. For the moment, the UTQL server uses a built-in heuristic to determine the quality of edges. In the following example Query only the best edge from A to B will be returned, even if there are multiple candidates

```
<Pattern name="SingleEdgeQuery">
  <Input>
    <Node name="A"/>
    <Node name="B"/>
    <Edge name="A2B" source="A" destination="B">
      <Predicate>type=='6D'</Predicate>
    </Edge>
  </Input>

  <Constraints>
    <OnlyBestEdgeMatch/>
  </Constraints>
</Pattern>
```

3.4 Description of `DataflowConfiguration` tags

The content of the `DataflowConfiguration` section in not specified in this document, as it is not parsed by the Ubitrack server, but sent back unmodified to the client when a data flow component needs to be instantiated. The section can be used by Ubitrack clients to store information about how to instantiate the data flow component. An OpenTracker client e.g. could insert the XML description of an OpenTracker node.

3.4.1 Description of `Attribute` tags

`Attribute` tags can have two different forms:

```
<Attribute name="name" value="someValue"/>
```

In this simple name-value form, a single string is stored as the attribute, which can be used as for predicate matching in `Input` sections.

```
<Attribute name="someName"><Value>...</Value></Attribute>
```

In this form, arbitrary XML-Data can be stored in the attribute with the specified name. The data, however, cannot be used for predicate matching in `Input` sections. This form is mostly useful to specify node- or edge-specific data for the configuration of data flow components.

4 Identifiers

UTQL uses names and identifiers to refer to attributes, nodes, edges and pattern instances in queries.

4.1 Scope

To distinguish between local (within a `Pattern`) and global names, UTQL knows both `name` and `id` attributes. All nodes and edges (both input and output) defined in a pattern must have a `name` attribute that is unique within this pattern. Ids, which are unique within the Ubitrack system, are used in two cases:

- All nodes defined in `Output` sections that have the same `id` are considered being the same.
- When a server returns a data flow description (see5), each pattern gets assigned an `id`, as each pattern can have multiple instances.

4.2 Predicate Syntax

The syntax of predicates used in `<Input>` sections is given in Backus-Naur form:

```
ATTR  ::= <attribute name> | <node/edge name>.<attribute name>
CONST ::= "'" <string> "'" | <number>
VAL   ::= ATTR | CONST
CMP   ::= "==" | "!=" | ">" | ">=" | "<" | "<="
STMT  ::= VAL CMP VAL | <external-predicate-name> "(" <args> ")"
PRED  ::= STMT | "!" PRED | "(" PRED ")" | PRED "&&" PRED | PRED "||" PRED
```

While most of the predicate syntax is straightforward, there are two particularities that deserve attention:

While `<attribute name>` refers to attributes of the edge/node the predicate is defined for, `<node/edge name>.<attribute name>` can be used to refer to attributes of other nodes and edges defined within the same `<Input>` section. This can e.g. be used to require equality of particular attributes of two nodes or edges.

External predicates are implemented by a plug-in mechanism in the Ubitrack server. They are used to specify additional constraints that cannot be derived by Boolean expressions over the attributes. The spatial indexing component, providing containment queries, will be integrated using this mechanism:

```
<Input>
  <Node name="lostSheep">
    <Predicate>
      objectType=='Sheep' && !containedIn( 'barn' )
    </Predicate>
  </Node>
</Input>
```

4.2.1 Attribute Expressions Syntax

Attributes in `<Output>` sections can be described using an `<AttributeExpression>`, which is evaluated when the pattern is instantiated and can refer to attributes of nodes or edges queried in the `<Input>` tag. The syntax of these expressions is again given in Backus-Naur form:

```
ATTR  ::= <node/edge name>.<attribute name>
CONST ::= "'" <string> "'" | <number>
VAL   ::= ATTR | CONST
BOP   ::= "+" | "-" | "*" | "/"
EXPR  ::= VAL | -EXPR | (EXPR) | EXPR BOP EXPR
```

As attributes are dynamically typed, the "+" operator either means numeric addition or string concatenation, depending on the context.

5 Ubitrack Server Responses

When the Ubitrack server instantiates a `<Pattern>` that has no `<Output>` section it is considered the result of a query. In this case, the server resolves all dependencies and sends the result (queries and patterns the result depends on) back to the client in one single `<UTQLResponse>` block. `UTQLResponse` elements that the server sends back to the client have the same basic structure as `UTQLRequest` elements, with the following differences:

- As a `<Pattern>` specified by the client can match multiple places in the SRG, a query/pattern can have multiple instances. Therefore each returned pattern has a unique `id` attribute which is used to refer to contained nodes/edges by other patterns.
- For all nodes in the `<Input>` section, the `<Predicate>` tags are removed and the attributes of the node are added as `<Attribute>` tags. Note that this step collects ALL attributes of all noded with the same id, no matter in which pattern they were specified.
- All edges in the `<Input>` section are replaced by references to edges in other `<Pattern>` blocks. These references should be used by the client to connect components in the data flow network. The syntax for edge references is:

  ```
  <Edge name="<local name>" pattern-ref="<pattern-id>"
    edge-ref="<edge-name>"/>
  ```

  where `pattern-ref` refers to the id of another pattern, and `edge-ref` to the name of an output edge within this pattern. Also, attributes from the output edge are propagated to the connected input edge and can be used e.g. for dataflow configuration.
- `<AttributeExpression>` tags in the `<Output>` section also are replaced by `<Attribute>` tags containing the evaluated expression.

Note that the UTQL design, and therefore also the Ubitrack server, makes no assumptions about how data flow components are instantiated

and connected. It is up to the client to interpret the results and create a data flow network.

An example of a valid answer to the query from section 3.1 (assuming the client knows how to instantiate SheepTrackers) is given by the following XML snippet:

```
<UTQLResponse>
<Pattern name="SheepInBarnQuery" id="6234527254">
  <Input>
    <Node name="Barn" id="Barn"/>
    <Node name="Sheep" id="Sheep">
      <Attribute name="objectType" value="sheep"/>
      <Attribute name="color" value="black"/>
    </Node>

    <Edge name="SheepPosition" source="Barn" destination="Sheep"
      pattern-ref="1069458" edge-ref="SheepPose"/>
  </Input>
</Pattern>

<Pattern name="TrackedSheep" id="1069458">
  <Output>
    <Node name="BarnNode" id="Barn"/>
    <Node name="SheepNode" id="Sheep">
      <Attribute name="objectType" value="sheep"/>
      <Attribute name="color" value="black"/>
    </Node>

    <Edge name="SheepPose" source="BarnNode" destination="SheepNode">
      <Attribute name="type" value="6D"/>
    </Edge>
  </Output>

  <DataflowConfiguration>
    <OpenTracker> <SheepTracker/> </OpenTracker>
  </DataflowConfiguration>
</Pattern>

</UTQLResponse>
```

Normally, the client would also specify a `<DataflowConfiguration>` in its query that contains the component where the client can get out the actual tracking data.

6 Examples

This example shows a complete conversation between an application and the Ubitrack server. The application specifies an SRG with three nodes: "Hmd→ART→Sheep" and requests the edge "Hmd→Sheep". This is a typical configuration in our lab, where the ART tracking system is used for augmentation in optical see-through HMD. The application also tells the server about the inversion and multiplication patterns.

Please note that the following example is not directly usable in combination with the Ubitrack library. It focuses on the nature of the UTQL and lacks some minor information in terms of node/edge attributes that would have to be added to make the example actually run.

6.1 SRG Specification

At first, a UTQL conversation with the server is started:

```
<UTQLRequest>
```

Now the client sends the information about the ART tracker node and its configurations. Although this information could also be included in the description of the individual tracked objects and the associated edges, in this example the tracker node has its own `<Pattern>` to avoid redundant definitions.

```
<Pattern name="Art">
  <Output>
    <Node name="Art" id="Art">
      <Attribute name="artPort" value="5000"/>
    </Node>
  </Output>
</Pattern>
```

Next, the information about the HMD is sent to the server:

```
<Pattern name="Hmd">
  <Input>
    <Node name="Art">
      <Predicate>id=='Art'</Predicate>
    </Node>
  </Input>

  <Output>
    <Node name="ArtHmd" id="ArtHmd"/>
```

```
    <Edge name="ArtToArtHmd" source="Art" destination="ArtHmd">
      <Attribute name="type" value="6D"/>
      <Attribute name="artBodyId" value="3"/>
    </Edge>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="ArtTracker"/>
  </DataflowConfiguration>
</Pattern>
```

In this case the client is not based on OpenTracker, but uses the Ubitrack Library data flow developed at TUM. Therefore, the `<DataflowConfiguration>` contains UbitrackLib-specific information.

In the next step the same information is specified for the Sheep object. Note that we define Hmd and Sheep in two different patterns. As all pattern descriptions are atomic, this way we can instantiate Sheep and Hmd tracking separately, should an application require only one.

```
<Pattern name="Sheep">
  <Input>
    <Node name="Art"><Predicate>id=='Art'</Predicate></Node>
  </Input>

  <Output>
    <Node name="ArtSheep" id="ArtSheep"/>
    <Edge name="ArtToArtSheep" source="Art" destination="ArtSheep">
      <Attribute name="type" value="6D"/>
      <Attribute name="artBodyId" value="7"/>
    </Edge>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="ArtTracker"/>
  </DataflowConfiguration>
</Pattern>
```

Now the SRG of the application is fully-specified.

6.2 Pattern specification

In the next step, the client needs to tell the server about the data flow operation it supports it its own data flow. For this, it is specifying the "Multiplication" and "Inversion" patterns as individual sub-graphs.

```
<Pattern name="Multiplication">
  <Input>
```

```
    <Node name="NodeA"/>
    <Node name="NodeB"/>
    <Node name="NodeC"/>
    <Edge name="AB" source="NodeA" destination="NodeB">
      <Predicate>type==’6D’</Predicate>
    </Edge>
    <Edge name="BC" source="NodeB" destination="NodeC">
      <Predicate>type==’6D’</Predicate>
    </Edge>
  </Input>

  <Output>
    <Edge name="AC" source="NodeA" destination="NodeC">
      <Attribute name="type" value="6D"/>
    </Edge>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="Multiplication"/>
  </DataflowConfiguration>
</Pattern>

<Pattern name="Inversion">
  <Input>
    <Node name="NodeA"/>
    <Node name="NodeB"/>
    <Edge name="AB" source="NodeA" destination="NodeB">
      <Predicate>type=’6D’</Predicate>
    </Edge>
  </Input>

  <Output>
    <Edge name="BA" source="NodeB" destination="NodeA">
      <Attribute name="type" value="6D"/>
    </Edge>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="Inversion"/>
  </DataflowConfiguration>
</Pattern>
```

Note how nodes without a predicate match any node. Therefore all edges of `"type=6D"` are selected as candidates for pattern application.

6.3 The Query

As the last step, the client specifies its query for the Hmd→Sheep edge.

```
<Pattern name="SheepQuery">
  <Input>
    <Node name="Hmd">
      <Predicate>id==’ArtHmd’</Predicate>
    </Node>
    <Node name="Sheep">
      <Predicate>id==’ArtSheep’</Predicate>
    </Node>

    <Edge name="TheEdge" source="Hmd" destination="Sheep">
      <Predicate>type==’6D’</Predicate>
    </Edge>
  </Input>

  <DataflowConfiguration>
    <UbitrackLib class="ApplicationPushSink"/>
  </DataflowConfiguration>

</Pattern>
```

The `<DataflowConfiguration>` section describes the final data flow component instantiated by the query. It is of type ApplicationPushSink, which forwards any data it gets to the application using a callback function.

Finally, the message to the server can be finished:

```
</UTQLRequest>
```

6.4 The Server's Response

Having the SRG, the SR Patterns and the application's query, the server can perform pattern detection to find the best data flow that fulfils the query. The result only consists of sub-graphs that the client sent to the server, but with predicates resolved, pattern and node ids assigned and input edges being connected to output edges of other sub-graphs. The client would react to this answer by instantiating and connecting the data flow components described by each `<Pattern>` tag. How exactly this is done is up to the client, and not described in this document.

```
<UTQLResponse>

    <Pattern name="Hmd" id="02295875">
        <Input>
            <Node name="Art" id="Art">
                <Attribute name="artPort" value="5000"/>
            </Node>
        </Input>
```

```
    <Output>
        <Node name="ArtHmd" id="ArtHmd"/>
        <Edge name="ArtToArtHmd" source="Art" destination="ArtHmd">
            <Attribute name="type" value="6D"/>
            <Attribute name="artBodyId" value="3"/>
        </Edge>
    </Output>
    <DataflowConfiguration>
        <UbitrackLib class="ArtTracker"/>
    </DataflowConfiguration>
</Pattern>

<Pattern name="Sheep" id="520981785">
    <Input>
        <Node name="Art" id="Art">
            <Attribute name="artPort" value="5000"/>
        </Node>
    </Input>
    <Output>
        <Node name="ArtSheep" id="ArtSheep"/>
        <Edge name="ArtToArtSheep" source="Art" destination="ArtSheep">
            <Attribute name="type" value="6D"/>
            <Attribute name="artBodyId" value="6D"/>
        </Edge>
    </Output>
    <DataflowConfiguration>
        <UbitrackLib class="ArtTracker"/>
    </DataflowConfiguration>
</Pattern>

<Pattern name="Multiplication" id="93759875">
    <Input>
        <Node name="NodeA" id="ArtHmd"/>
        <Node name="NodeB" id="Art">
            <Attribute name="artPort" value="5000"/>
        </Node>
        <Node name="NodeC" id="ArtSheep"/>
        <Edge name="AB" source="NodeA" destination="NodeB"
              pattern-ref="16987234" edge-ref="BA">
            <Attribute name="type" value="6D"/>
        </Edge>
        <Edge name="BC" source="NodeB" destination="NodeC"
              pattern-ref="520981785" edge-ref="ArtToArtSheep">
            <Attribute name="type" value="6D"/>
        </Edge>
    </Input>
    <Output>
        <Edge name="AC" source="NodeA" destination="NodeC">
            <Attribute name="type" value="6D"/>
```

```
            </Edge>
        </Output>
        <DataflowConfiguration>
            <UbitrackLib class="Multiplication"/>
        </DataflowConfiguration>
    </Pattern>

    <Pattern name="Inversion" id="16987234">
        <Input>
            <Node name="NodeA" id="Art">
                <Attribute name="artPort" value="5000"/>
            </Node>
            <Node name="NodeB" id="ArtHmd"/>
            <Edge name="AB" source="NodeA" destination="NodeB"
                  pattern-ref="02295875" edge-ref="ArtToArtHmd">
                <Attribute name="type" value="6D"/>
            </Edge>
        </Input>
        <Output>
            <Edge name="BA" source="NodeB" destination="NodeA">
                <Attribute name="type" value="6D"/>
            </Edge>
        </Output>
        <DataflowConfiguration>
            <UbitrackLib class="Inversion"/>
        </DataflowConfiguration>
    </Pattern>

    <Pattern name="SheepQuery" id="2347545">
        <Input>
            <Node name="Hmd" id="ArtHmd"/>
            <Node name="Sheep" id="ArtSheep"/>
            <Edge name="TheEdge" source="Hmd" destination="Sheep"
                  pattern-ref="93759875" edge-ref="AC">
                <Attribute name="type" value="6D"/>
            </Edge>
        </Input>
        <DataflowConfiguration>
            <UbitrackLib class="ApplicationPushSink"/>
        </DataflowConfiguration>
    </Pattern>

</UTQLResponse>
```

6.4.1 Removing the Query

If the client is no longer interested, it can remove the query by replacing it with an empty one.

```
<UTQLRequest>
  <Pattern name="SheepQuery"/>
</UTQLRequest>
```

Now the server should respond by removing all the data flow components on the client's side. Note that, as long as the connection to the server is up, the patterns and the SRG still remain in the server database and can be instantiated again, if other applications require it.

7 Pattern Documentation

UTQL also allows patterns, nodes, edges and attributes to provide documentation strings by including an `<Description>` element. This way, users can be provided with an explanation of the meaning of patterns and their parameters. Note that the `<Description>` elements can include arbitrary HTML-formatted strings. The `xmlns="http://www.w3.org/1999/xhtml` attribute is only necessary if schema validation is desired. See A for further information.

For graphical UTQL front-ends, `Pattern`, `Edge`, `Node` and `Attribute` elements can optionally include a `displayName` attribute that is presented to the user instead of the real name, which sometimes can be cryptic.

```
<Pattern name="PosePushPullMultiplication"
         displayName="Push-Pull Multiplication of Pose">
    <Description>
        <p xmlns="http://www.w3.org/1999/xhtml">
            This pattern multiplies two poses <tt
            xmlns="http://www.w3.org/1999/xhtml">AB</tt> and <tt
            xmlns="http://www.w3.org/1999/xhtml">BC</tt>, where <tt
            xmlns="http://www.w3.org/1999/xhtml">AB</tt> is provided as a push
            input, and <tt xmlns="http://www.w3.org/1999/xhtml">BC</tt> as
            pull. The result is pushed.
        </p>
    </Description>
    <Input>
        <Node name="A" displayName="source node">
            <Description>
                <p xmlns="http://www.w3.org/1999/xhtml">
                    The source node of <tt
        xmlns="http://www.w3.org/1999/xhtml>AB</tt> and the resulting
        transformation
                </p>
            </Description>
        </Node>
        <Node name="B" displayName="intermediate node">
```

```
        <Description>
            <p xmlns="http://www.w3.org/1999/xhtml">
                The source node of <tt
    xmlns="http://www.w3.org/1999/xhtml">BC</tt> and destination of <tt
    xmlns="http://www.w3.org/1999/xhtml">AB</tt>
            </p>
        </Description>
    </Node>
    <Node name="C" displayName="destination node">
        <Description>
            <p xmlns="http://www.w3.org/1999/xhtml">
                The destination node of <tt
    xmlns="http://www.w3.org/1999/xhtml">BC</tt> and the resulting
    transformation
            </p>
        </Description>
    </Node>
    <Edge name="AB" displayName="AB" source="A" destination="B">
        <Description>
            <p xmlns="http://www.w3.org/1999/xhtml">
                The transformation <tt
                xmlns="http://www.w3.org/1999/xhtml">AB</tt>as push
            </p>
        </Description>
        <Predicate>&amp;amp;amp;</Predicate>
    </Edge>
    <Edge name="BC" displayName="BC" source="B" destination="C">
        <Description>
            <p xmlns="http://www.w3.org/1999/xhtml">
                The transformation <tt
                xmlns="http://www.w3.org/1999/xhtml">BC</tt>as pull
            </p>
        </Description>
        <Predicate>&amp;amp;amp;</Predicate>
    </Edge>
</Input>
<Output>
    <Edge name="AC" displayName="AC (result)" source="A" destination="C">
        <Description>
            <p xmlns="http://www.w3.org/1999/xhtml">
                Resulting transformation <tt
                xmlns="http://www.w3.org/1999/xhtml">AC</tt>as push
            </p>
        </Description>
        <Attribute name="type" value="6D" displayName="data type">
            <Description>
                <p xmlns="http://www.w3.org/1999/xhtml">
                    Data type of the resulting transformation
                </p>
```

```
            </Description>
        </Attribute>
        <Attribute name="mode" value="push" displayName="push/pull mode">
            <Description>
                <p xmlns="http://www.w3.org/1999/xhtml">
                    The result is output as push
                </p>
            </Description>
        </Attribute>
    </Edge>
</Output>
</Pattern>
```

A UTQL XSD Schema

In the following, a W3C XML Schema is given which describes valid UTQL expressions. It covers both communication directions, UTQL requests (SRG-/pattern specification and queries) being sent from an Ubitrack client to the Ubitrack server as well as UTQL responses being returned by the Ubitrack server to connected clients.

The schema incorporates as much semantical checks as possible. There is only requirement defined in this document that cannot be checked in this way automatically: in a UTQL response, it cannot be guaranteed that an output edge referenced by an input edge (via the `pattern-ref` and `edge-ref` attributes) really exists. This is a restriction imposed by the XML Schema language.

If you want to use the schema to parse or validate your UTQL documents, use the following stub to wrap your patterns. Replace `<UTQLRequest>` by `<UTQLResponse>` for dataflow descriptions generated by the Ubitrack server.

```
<?xml version="1.0" encoding="UTF-8"?>

<UTQLRequest xmlns='http://ar.in.tum.de/ubitrack/utql'
                      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                      xmlns:xi="http://www.w3.org/2001/XInclude"
                      xsi:schemaLocation='http://ar.in.tum.de/ubitrack/utql
                      http://ar.in.tum.de/files/ubitrack/utql/utql.xsd
                      http://www.w3.org/1999/xhtml
                      http://ar.in.tum.de/files/ubitrack/utql/xhtml1-strict.xsd'>

</UTQLRequest>
```

The schema itself is presented in the following. The most recent version is available via http://ar.in.tum.de/files/ubitrack/utql/utql.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : utql.xsd
    Created on : April 27, 2007, 10:45 AM
    Author     : Peter Keitler
    Description: See annotations of this schema
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:utql="http://ar.in.tum.de/ubitrack/utql"
        xmlns:xml="http://www.w3.org/XML/1998/namespace"
        xmlns:xi="http://www.w3.org/2001/XInclude"
        targetNamespace="http://ar.in.tum.de/ubitrack/utql"
        elementFormDefault="qualified"
        xml:lang="en-GB">

    <annotation>
    <documentation>Schema describing the Ubiquitous Tracking Query Language
    (UTQL) for communication of Ubitrack clients and server. The schema covers
    both communication directions, thus application queries and the server's
    response.</documentation>
    </annotation>

    <import namespace="http://www.w3.org/2001/XInclude"
    schemaLocation="http://www.w3.org/2001/XInclude.xsd"/>



    <element name="UTQLRequest" type="utql:UTQLRequestType">
        <annotation>
            <documentation>Root element of a UTQL client request. It contains at
            least one pattern element</documentation>
        </annotation>
    </element>

    <element name="UTQLResponse" type="utql:UTQLResponseType">
        <annotation>
            <documentation>Root element of a UTQL server response. It contains
            at least one pattern element.</documentation>
        </annotation>

        <key name="ResponsePatternIDKey">
            <annotation>
                <documentation>Ensure that the id of all pattern elements is
                unique</documentation>
            </annotation>

            <selector xpath="utql:Pattern"/>
            <field xpath="@id"/>
```

```
    </key>

    <keyref name="ResponsePatternIDKeyRef"
        refer="utql:ResponsePatternIDKey">
        <annotation>
            <documentation>Ensure that all pattern elements referred by Edge
            elements belonging to some other Input element
            exist.</documentation>
        </annotation>

        <selector xpath="utql:Pattern/utql:Input/utql:Edge"/>
        <field xpath="@pattern-ref"/>
    </keyref>
</element>



<complexType name="UTQLRequestType">
    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
        <element name="Pattern" type="utql:RequestPatternType"
    maxOccurs="unbounded">

            <key name="QueryEdgeNameKey">
                <annotation>
                    <documentation>Ensure that each Edge element belonging
                    to this pattern has a unique name.</documentation>
                </annotation>

                <selector
        xpath="utql:Input/utql:Edge|utql:Output/utql:Edge"/>
                <field xpath="@name"/>
            </key>
            <key name="QueryInputEdgeNameKey">
                <annotation>
                    <documentation>Ensure that each Edge element belonging
                    to the Input element of this pattern has a unique
                    name. This is redundant because key QueryEdgeNameKey
                    also includes the edge referred here, but we need this
                    key nevertheless below in order to ensure that edges
                    referenced in the Constraints element really exist in
                    the Input element of this pattern.</documentation>
                </annotation>

                <selector
        xpath="utql:Input/utql:Edge|utql:Output/utql:Edge"/>
                <field xpath="@name"/>
            </key>
```

```
            <key name="QueryNodeNameKey">
                <annotation>
                    <documentation>Ensure that each Node element belonging
                    to this pattern has a unique name.</documentation>
                </annotation>

                <selector
      xpath="utql:Input/utql:Node|utql:Output/utql:Node"/>
                <field xpath="@name"/>
            </key>
            <key name="QueryInputNodeNameKey">
                <annotation>
                    <documentation>Ensure that each Node element belonging
                    to the Input element of this pattern has a unique
                    name. This is redundant because key QueryNodeNameKey
                    also includes the nodes referred here, but we need this
                    key nevertheless in order to ensure that edges in the
                    Input element only refer Nodes from the Input element as
                    their source or destination and that edges referenced in
                    the Constraints element really exist in the Input
                    element of this pattern. See also below.</documentation>
                </annotation>

                <selector xpath="utql:Input/utql:Node"/>
                <field xpath="@name"/>
            </key>
  <keyref name="QueryInputSourceNodeNameKeyRef"
    refer="utql:QueryInputNodeNameKey">
    <annotation>
                    <documentation>Ensure that all Node elements referred by
                    Edge elements belonging to the Input element of this
                    pattern as their source node, exist in this Input
                    element, too.</documentation>
                </annotation>

                <selector xpath="utql:Input/utql:Edge"/>
                <field xpath="@source"/>
            </keyref>
            <keyref name="QueryInputDestNodeNameKeyRef"
                refer="utql:QueryInputNodeNameKey">
                <annotation>
                    <documentation>Ensure that all Node elements referred by
                    Edge elements belonging to the Input element of this
                    pattern as their destination node, exist in this Input
                    element, too.</documentation>
                </annotation>

                <selector xpath="utql:Input/utql:Edge"/>
                <field xpath="@destination"/>
```

```
    </keyref>
    <keyref name="QueryOutputSourceNodeNameKeyRef"
        refer="utql:QueryNodeNameKey">
        <annotation>
            <documentation>Ensure that all Node elements referred by
            Edge elements belonging to this pattern as their source
            node, exist.</documentation>
        </annotation>

        <selector xpath="utql:Output/utql:Edge"/>
        <field xpath="@source"/>
    </keyref>
    <keyref name="QueryOutputDestNodeNameKeyRef"
        refer="utql:QueryNodeNameKey">
        <annotation>
            <documentation>Ensure that all Node elements referred by
            Edge elements belonging to this pattern as their
            destination node, exist.</documentation>
        </annotation>

        <selector xpath="utql:Output/utql:Edge"/>
        <field xpath="@destination"/>
    </keyref>
    <key name="ConstraintsIDKey">
        <annotation>
            <documentation>Ensure that all Correspondence elements
            have a unique name.</documentation>
        </annotation>

        <selector xpath="utql:Constraints/utql:Correspondence"/>
        <field xpath="@name"/>
    </key>
    <key name="ConstraintsNodeKey">
        <annotation>
            <documentation>Ensure that all nodes within a certain
            Correspondence element have a unique node
            reference.</documentation>
        </annotation>

        <selector
xpath="utql:Constraints/utql:Correspondence/utql:Node"/>
        <field xpath="@node-ref"/>
    </key>
    <keyref name="ConstraintsNodeKeyRef"
        refer="utql:QueryInputNodeNameKey">
        <annotation>
            <documentation>Ensure that all nodes referenced from
            within a Correspondence element really exist in the
            input section of this pattern.</documentation>
```

```
                </annotation>

                <selector
            xpath="utql:Constraints/utql:Correspondence/utql:Node"/>
                <field xpath="@node-ref"/>
            </keyref>
            <key name="ConstraintsEdgeKey">
                <annotation>
                    <documentation>Ensure that all edges within a certain
                    Correspondence element have a unique edge
                    reference.</documentation>
                </annotation>

                <selector
            xpath="utql:Constraints/utql:Correspondence/utql:Edge"/>
                <field xpath="@edge-ref"/>
            </key>
            <keyref name="ConstraintsEdgeKeyRef"
                refer="utql:QueryInputEdgeNameKey">
                <annotation>
                    <documentation>Ensure that all edge referenced from
                    within a Correspondence element really exist in the
                    input section of this pattern.</documentation>
                </annotation>

                <selector
            xpath="utql:Constraints/utql:Correspondence/utql:Edge"/>
                <field xpath="@edge-ref"/>
            </keyref>
        </element>
    </sequence>
    <attribute name="name" type="utql:IDType"/>
</complexType>

<complexType name="UTQLResponseType">
    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
        <element name="Pattern" type="utql:ResponsePatternType"
    maxOccurs="unbounded">
            <key name="EdgeNameKey">
                <annotation>
                    <documentation>Ensure that each Edge element belonging
                    to this pattern has a unique name.</documentation>
                </annotation>

                <selector
            xpath="utql:Input/utql:Edge|utql:Output/utql:Edge"/>
                <field xpath="@name"/>
```

```
    </key>

    <key name="NodeNameKey">
        <annotation>
            <documentation>Ensure that each Node element belonging
            to this pattern has a unique name.</documentation>
        </annotation>

        <selector
xpath="utql:Input/utql:Node|utql:Output/utql:Node"/>
        <field xpath="@name"/>
    </key>
    <key name="InputNodeNameKey">
        <annotation>
            <documentation>Ensure that each Node element belonging
            to the Input element of this pattern has a unique
            name. This is redundant because key NodeNameKey also
            includes the nodes referred here, but we need this key
            nevertheless it in order to ensure that edges in the
            Input element only refer Nodes from the Input element as
            their source or destination. See also
            below.</documentation>
        </annotation>

        <selector xpath="utql:Input/utql:Node"/>
        <field xpath="@name"/>
    </key>
    <keyref name="InputSourceNodeNameKeyRef"
        refer="utql:InputNodeNameKey">
        <annotation>
            <documentation>Ensure that all Node elements referred by
            Edge elements belonging to the Input element of this
            pattern as their source node, exist in this Input
            element, too.</documentation>
        </annotation>

        <selector xpath="utql:Input/utql:Edge"/>
        <field xpath="@source"/>
    </keyref>
    <keyref name="InputDestNodeNameKeyRef"
        refer="utql:InputNodeNameKey">
        <annotation>
            <documentation>Ensure that all Node elements referred by
            Edge elements belonging to the Input element of this
            pattern as their destination node, exist in this Input
            element, too.</documentation>
        </annotation>

        <selector xpath="utql:Input/utql:Edge"/>
```

```
                <field xpath="@destination"/>
            </keyref>
            <keyref name="OutputSourceNodeNameKeyRef"
                refer="utql:NodeNameKey">
                <annotation>
                    <documentation>Ensure that all Node elements referred by
                    Edge elements belonging to this pattern as their source
                    node, exist.</documentation>
                </annotation>

                <selector xpath="utql:Output/utql:Edge"/>
                <field xpath="@source"/>
            </keyref>
            <keyref name="OutputDestNodeNameKeyRef"
                refer="utql:NodeNameKey">
                <annotation>
                    <documentation>Ensure that all Node elements referred by
                    Edge elements belonging to this pattern as their
                    destination node, exist.</documentation>
                </annotation>

                <selector xpath="utql:Output/utql:Edge"/>
                <field xpath="@destination"/>
            </keyref>
        </element>
    </sequence>
    <attribute name="name" type="utql:IDType"/>
</complexType>



<complexType name="AbstractPatternType">
    <annotation>
        <documentation>Base type for both types of patterns, request
        patterns and response patterns</documentation>
    </annotation>

    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
    </sequence>
    <attribute name="name" type="utql:IDType" use="required"/>
    <attribute name="displayName" type="string"/>
</complexType>

<complexType name="RequestPatternType">
    <annotation>
        <documentation>Each query pattern has a name attribute and
        optionally, an id attribute which may be used for reconstruction of
```

```
        edge dependencies without actually evaluating the edge predicate. It
        contains at most one Input element, at most one Output element and
        at most one dataflow configuration element of each type
        (e.g. DataflowConfiguration and/or
        UbitrackDataflowConfiguration).</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractPatternType">
            <sequence minOccurs="1">
                <element minOccurs="0" name="Input"
                    type="utql:RequestInputType"/>
                <element minOccurs="0" name="Output"
        type="utql:RequestOutputType"/>
                <element minOccurs="0" name="Constraints"
        type="utql:ConstraintsType"/>
                <element minOccurs="0" name="DataflowConfiguration"
        type="utql:DataflowConfigurationType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ResponsePatternType">
    <annotation>
        <documentation>Each query pattern has a name attribute and
        optionally, an id attribute which may be used for reconstruction of
        edge dependencies without actually evaluating the edge predicate. It
        contains at most one Input element, at most one Output element and
        at most one dataflow configuration element of each type
        (e.g. DataflowConfiguration and/or
        UbitrackDataflowConfiguration).</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractPatternType">
            <sequence minOccurs="1">
                <element minOccurs="0" name="Input"
        type="utql:ResponseInputType"/>
                <element minOccurs="0" name="Output"
        type="utql:ResponseOutputType"/>
                <element minOccurs="0" name="DataflowConfiguration"
        type="utql:DataflowConfigurationType"/>
            </sequence>
            <attribute name="idd" type="utql:IDType"/>
        </extension>
    </complexContent>
</complexType>
```

```
<complexType name="RequestInputType">
    <annotation>
        <documentation>The Input element consists of at least one Node
        element of type QueryNodeType, followed by an arbitrary number of
        Edge elements followed by QueryEdgeType.</documentation>
    </annotation>

    <sequence>
        <element name="Node" maxOccurs="unbounded"
    type="utql:RequestQueryNodeType"/>
        <element name="Edge" maxOccurs="unbounded" minOccurs="0"
    type="utql:QueryEdgeType"/>
    </sequence>
</complexType>

<complexType name="RequestOutputType">
    <annotation>
        <documentation>The Output element consists of at least one Node
        element, followed by an arbitrary number of Edge
        elements.</documentation>
    </annotation>

    <sequence>
        <element name="Node" maxOccurs="unbounded" minOccurs="0"
    type="utql:RequestOutputNodeType"/>
        <element name="Edge" maxOccurs="unbounded" minOccurs="0"
    type="utql:OutputEdgeType"/>
    </sequence>
</complexType>

<complexType name="ResponseInputType">
    <annotation>
        <documentation>The Input element consists of at least one Node
        element, followed by an arbitrary number of Edge elements. Both
        nodes and edges are already resolved, thus do not contain Predicate
        elements. See ResolvedNodeType and ResolvedEdgeType for further
        information.</documentation>
    </annotation>

    <sequence>
        <element name="Node" maxOccurs="unbounded"
    type="utql:ResponseInputNodeType"/>
        <element name="Edge" maxOccurs="unbounded" minOccurs="0"
    type="utql:ResolvedEdgeType"/>
    </sequence>
</complexType>
```

```
<complexType name="ResponseOutputType">
    <annotation>
        <documentation>The Output element consists of an arbitrary number
        Node elements, followed by an arbitrary number of Edge
        elements.</documentation>
    </annotation>

    <sequence>
        <element name="Node" maxOccurs="unbounded" minOccurs="0"
    type="utql:ResponseOutputNodeType"/>
        <element name="Edge" maxOccurs="unbounded" minOccurs="0"
    type="utql:OutputEdgeType"/>
    </sequence>
</complexType>



<complexType name="AbstractNodeType" abstract="true">
    <annotation>
        <documentation>Abstract base type for all Node elements. Each node
        has a name attribute that is unique throughout the pattern it
        belongs to.</documentation>
    </annotation>

    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
        <element name="GuiPos" minOccurs="0" type="utql:GuiCoordType"/>
    </sequence>
    <attribute name="name" type="utql:IDType" use="required"/>
    <attribute name="displayName" type="string"/>
</complexType>

<complexType name="RequestQueryNodeType">
    <annotation>
        <documentation>Node type used for querying nodes already available
        in the world SRG, contained in the UTQLRequest/Pattern/Input
        element. In addition to the features of all nodes, it contains a
        sequence of Predicate elements that restrict the matching of this
        node against other nodes contained in the Output element of
        previously registered patterns.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractNodeType">
            <sequence>
                <element name="Predicate" minOccurs="0" maxOccurs="1"
                type="string"/>
            </sequence>
```

```
        </extension>
    </complexContent>
</complexType>

<complexType name="RequestOutputNodeType">
    <annotation>
        <documentation>Node type used registering new nodes in the world
        SRG, contained in the UTQLRequest/Pattern/Output element. In
        addition to the features of all nodes, it is being assigned a
        globally unique ID and a list of attributes. In case more than one
        nodes with the same ID are registered, those nodes will be
        considered to be the same and their attributes will be
        merged.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractNodeType">
            <choice maxOccurs="unbounded">
                <element name="Attribute" minOccurs="0"
                maxOccurs="unbounded" type="utql:AttributeType"/>
            </choice>
            <attribute name="id" type="utql:IDType"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="AbstractResponseNodeType" abstract="true">
    <annotation>
        <documentation>Abstract base type of Node elements used in
        UTQLResponse elements. Each such node has a list of
        attributes.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractNodeType">
            <sequence>
                <element name="Attribute" minOccurs="0"
        maxOccurs="unbounded" type="utql:AttributeType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ResponseInputNodeType">
    <annotation>
        <documentation>Each such Node has a globally unique id attribute
        which references a particular Node of type ResponseOutputNodeType
        (corresponding to one particular node in the world SRG) which also
        has to be contained in the same UTQLResponse and carry the same
```

```
        id. Nodes of type RequestQueryNodeType that matches a Node of Type
        RequestOutputNodeType in a UTQLRequest becomes a Node of
        ResponseInputNodeType. In one UTQLResponse, there may be one or more
        Node elements of type RequestInputNodeType that refer to one single
        Node of type ResponseOutputNodeType.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractResponseNodeType">
            <attribute name="id" type="utql:IDType" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ResponseOutputNodeType">
    <annotation>
        <documentation>Each such Node has a globally unique id and
        corresponds to exactly one node in the world SRG. There may be only
        one Node of this type with a certain id.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractResponseNodeType">
            <attribute name="id" type="utql:IDType"/>
        </extension>
    </complexContent>
</complexType>



<complexType name="AbstractEdgeType" abstract="true">
    <annotation>
        <documentation>An edge has a mandatory name attribute that is unique
        throughout the Pattern it belongs to. The source and destination
        attributes denote Node elements contained in either the Input or the
        Output element belonging to the pattern element in which this Edge
        is contained.</documentation>
    </annotation>

    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
  <element name="GuiLandmark" type="utql:GuiCoordType" minOccurs="0"/>
    <element name="GuiLabelPos" type="utql:GuiCoordType" minOccurs="0"/>
    </sequence>
    <attribute name="name" type="utql:IDType" use="required"/>
    <attribute name="source" type="utql:IDType" use="required"/>
    <attribute name="destination" type="utql:IDType" use="required"/>
    <attribute name="displayName" type="string"/>
```

```
</complexType>

<complexType name="QueryEdgeType">
    <annotation>
        <documentation> An edge query element contained in an Input element,
        in addition to the properties shared with all edges, contains a
        sequence of Predicate elements that restrict matching of this edge
        against other edges contained in the Output element of previously
        registered subgtraphs. This restricts the characteristics of
        potential dataflow networks.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractEdgeType">
            <sequence>
                <element name="Predicate" minOccurs="0" maxOccurs="1"
  type="string"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ResolvedEdgeType">
    <annotation>
        <documentation>An edge reference element contained in an Input
        element. In addition to the properties shared with all edges, it
        contains the pattern-ref and edge-ref attribute which refer to
        another pattern and the corresponding edge contained in the
        referenced pattern's Output element. This defines the dependencies
        of the dataflow network. A match of an Edge of type QueryEdgeType
        may become an edge of type ResolvedEdgeType.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractEdgeType">
            <sequence>
                <element name="Attribute" minOccurs="0"
  maxOccurs="unbounded" type="utql:AttributeType"/>
            </sequence>
            <attribute name="pattern-ref" type="utql:IDType"
      use="required"/>
            <attribute name="edge-ref" type="utql:IDType" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="OutputEdgeType">
    <annotation>
        <documentation>An Edge element contained in an Ouput element, in
```

```
        addition to the properties shared with all edges, may contain an
        arbitrary number of Attribute elements.</documentation>
    </annotation>

    <complexContent>
        <extension base="utql:AbstractEdgeType">
            <choice maxOccurs="unbounded">
                <element name="Attribute" minOccurs="0"
        maxOccurs="unbounded" type="utql:AttributeType"/>
            </choice>
        </extension>
    </complexContent>
</complexType>



<complexType name="ConstraintsType">
    <annotation>
        <documentation>Information that constrains the degrees of freedom in
        matching the inuput section of a pattern.</documentation>
    </annotation>

    <sequence>
        <element name="Correspondence" minOccurs="0" maxOccurs="unbounded">
            <complexType>
                <sequence>
                    <element name="Node" minOccurs="0"
          maxOccurs="unbounded">
                        <complexType>
                            <attribute name="node-ref" type="utql:IDType"/>
                        </complexType>
                    </element>
                    <element name="Edge" minOccurs="0"
          maxOccurs="unbounded">
                        <complexType>
                            <attribute name="edge-ref" type="utql:IDType"/>
                        </complexType>
                    </element>
                </sequence>
                <attribute name="name" type="utql:IDType"/>
                <attribute name="minMultiplicity" type="integer"/>
                <attribute name="maxMultiplicity" type="integer"/>
                <attribute name="stepSize" type="integer"/>
            </complexType>
        </element>
        <element name="OnlyBestEdgeMatch" minOccurs="0">
            <complexType>
            </complexType>
        </element>
```

```
    </sequence>
</complexType>



<complexType name="DescriptionType">
    <annotation>
        <documentation>Defines the layout of an element's description in
        arbitrary XHTML. Once a valid XHTML XML schema is provided at
        http://www.w3.org/1999/xhtml.xsd (currently, only DTD available),
        processContents can be switched to strict. </documentation>
    </annotation>

    <sequence>
        <any namespace="http://www.w3.org/1999/xhtml" minOccurs="1"
    maxOccurs="unbounded" processContents="strict"/>
    </sequence>
</complexType>



<complexType name="AttributeType">
    <annotation>
        <documentation>Generic attribute type with an attribute name and the
        corresponding value. The latter can be set directly on the optional
        value attribute (for short strings) or as arbitrary XML
        content.</documentation>
    </annotation>

    <sequence>
        <element name="Description" minOccurs="0"
    type="utql:DescriptionType"/>
        <element name="Value" minOccurs="0"
    type="utql:ArbitraryXmlValueType"/>
    </sequence>
    <attribute name="name" type="string" use="required"/>
    <attribute name="displayName" type="string"/>
    <attribute name="value" type="string"/>
</complexType>

<complexType name="ArbitraryXmlValueType">
    <complexContent mixed="true">
        <restriction base="anyType">
            <sequence>
                <any processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
            </sequence>
        </restriction>
    </complexContent>
```

```
    </complexType>


    <simpleType name="IDType">
        <annotation>
            <documentation>Type which ensures that an ID is always
            valid.</documentation>
        </annotation>

        <restriction base="string">
            <minLength value="1"/>
        </restriction>
    </simpleType>



    <complexType name="GuiCoordType">
        <annotation>
            <documentation>Type for storing display coordinates, used for
            storing the positions of nodes and edges.</documentation>
        </annotation>

        <attribute name="x" type="short" use="required"/>
        <attribute name="y" type="short" use="required"/>
    </complexType>



    <complexType name="DataflowConfigurationType">
        <sequence>
            <element name="UbitrackLib">
                <complexType>
                    <attribute name="class" type="utql:IDType"/>
                </complexType>
            </element>
            <choice maxOccurs="unbounded">
                <element name="Attribute" minOccurs="0" maxOccurs="unbounded"
          type="utql:AttributeType"/>
            </choice>
        </sequence>
    </complexType>
</schema>
```