# Generating Query Forms and Reports for Semistructured Data: the QURSED Editor

Yannis Papakonstantinou [1], Michalis Petropoulos [1], and Vasilis Vassalos [2]

[1] Computer Science and Engineering Department
University of California, San Diego, USA
Email: {yannis, mpetropo}@cs.ucsd.edu
[2] Computer Science Department, Athens University of Economics and Business
Athens, Greece
Email: vassalos@aueb.gr

**Abstract.** The wide adoption of semistructured XML databases requires the existence of systems for the generation and execution of web-based interactive database query forms and reports. Such systems are most effective when they allow the construction of the query forms and reports without programming, via the use of intuitive graphical tools. We describe the architecture of the QURSED system for the declarative specification and automatic generation of web-based query forms and reports (QFRs) for semistructured XML data. We then focus on the QURSED Editor, a powerful GUI tool for the generation of the declarative specifications of QFRs. We describe the Editor's architecture and present the techniques and heuristics the Editor employs for translating visual designer input into meaningful specifications of query forms and reports. An on-line demonstration of the system is available at http://www.db.ucsd.edu/qursed

## 1. Introduction

XML is a simple and powerful data exchange and representation language, largely due to its self-describing nature. Its advantages are especially strong in the case of semistructured data, i.e., data whose structure is not rigid and is characterized by nesting, optional fields, and high variability of the structure. An example is a catalog for complicated products such as sensors: they are often nested into manufacturer categories and each product of a sensor manufacturer comes with its own variations. For example, some sensors are rectangular and have height and width, and others are cylindrical and have diameter and barrel style. Some sensors have one or more protection ratings, while others have none. The relational data model is cumbersome in modeling such semistructured data because of its rigid tabular structure.

The database community perceived the relational model's limitations early on and responded with labeled graph data models [1] that evolved into XML-based data models [10]. XML query languages (with most notable the emerging XQuery standard [6]), XML databases [23] and mediators [12][17] have been designed and developed. They materialize the in-principle advantages of XML in representing and querying semistructured data. QURSED automates the construction of web-based query forms and reports for querying semistructured, XML data.

Web-based query forms and reports are an important aspect of real-world database systems [4], albeit semi-neglected by the database research community. They allow millions of web users to selectively view the information of underlying sources. A number of commercial tools, such as Macromedia Dreamweaver Ultradev, Macromedia Coldfusion, and Microsoft Visual Interdev, facilitate the development of web-based query forms and reports that access relational databases. However, these tools are tied to the relational model, which limits the resulting user experience and impedes the developer in his efforts to quickly and cleanly produce web-based query forms and reports. We present the QURSED Editor that is, to the best of our knowledge, the first web-based query forms and reports tool for semistructured XML data.

The Editor is a powerful GUI tool that takes visual input by a form designer and produces declarative specifications of query form and report pages that are called *Query Set Specifications* (*QSS*). A *QSS* describes formally the complex query and reporting *capabilities* [24] of a query form. These capabilities include the large number of queries that a form can generate to the underlying XML query processor and the different structure and content of the query result. The Editor allows the translation of visual input into meaningful Query Set Specifications. The next section describes the QURSED system architecture, the process and the actions involved in producing a *QFR*, and the process by which a *QFR* interacts with the end-user, emits a query and displays the result. We also introduce terms used in the rest of the paper.

## 1.1 System Overview and Architecture

The QURSED system architecture is shown in Figure 1. QURSED consists of the *QURSED Editor*, which is the design-time component and the main focus of this paper, the *QURSED Compiler,* and the *QURSED Run Time Engine*. A detailed description of QURSED, including important features omitted from this overview, can be found in [19].

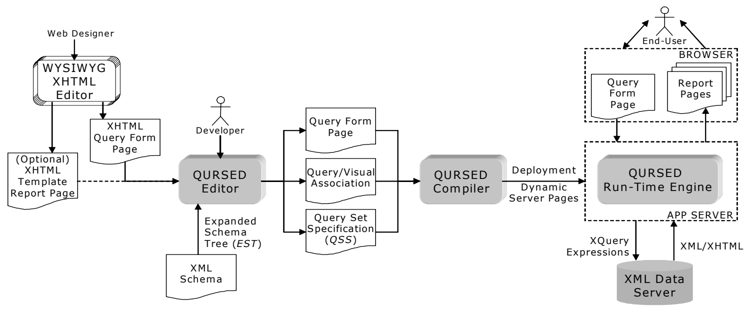


**Figure 1** QURSED System Architecture

The Editor inputs the XML Schema that describes the structure of the XML data to be queried and constructs an *Expanded Schema Tree* (*EST*) out of it. The *EST* is a visual abstraction of the XML Schema that the developer interacts with. The Editor also inputs an *XHTML query form page* that provides the (XHTML) static part of the form page, including the XHTML form controls [22], such as `select` ("drop-down menus") and `text` ("fill-in-the-box") input controls, that the end-user will be interacting with. It may additionally input an optional *template report page* that provides the XHTML structure of the report page. In particular, it depicts the nested tables and other components of the page. It is just a template, since we may not know in advance how many rows/tuples appear in each table. The query form and template report pages are typically developed with an external “What You See Is What You Get” (WYSIWYG) editor, such as Macromedia HomeSite. If a template report page is not provided, the developer can build one using the Editor.

The Editor displays the *EST* and the XHTML pages to the developer, who uses them to build the Query Set Specification *(QSS)* of the *QFR* and the *query/visual association*. The specification focuses on the query capabilities of the *QFR* and describes the set of queries that the form may emit. It includes *condition fragments* and the *result tree*. Each condition fragment stands for a set of conditions (typically navigations, selections and joins) that contain *parameters*.

The query/visual association indicates how each parameter is associated with corresponding *XHTML form controls* [22] of the query form page. The form controls that are associated with the parameters contained in a condition fragment constitute its *visual fragment*. Finally, the result tree specifies how the source data instantiate and populate the XHTML template report page.

The *QURSED Compiler* takes as input the output of the Editor and produces *dynamic server pages*, which control the interaction with the end-user. The dynamic server pages, the query set specification, and the query/visual association are inputs to the *QURSED Run-time Engine*. The dynamic server pages handle the navigation on the report page. The engine, based on the query set specification and the query/visual association, generates an XQuery expression when the end-user clicks “Execute”, which is sent to the XML Data Server. The query results are expressed directly in XHTML and are processed again by QURSED to generate the report pages.

The rest of the paper is organized as follows. Related work and the contributions of the QURSED Editor are presented in Section 2. In Section 3 the running example is introduced and the necessary QURSED system internals are presented, to allow for the presentation of the Editor. Section 4 presents the Editor, including the architecture, a description of the visual actions it supports, and the way they are translated into meaningful query set specifications and query/visual associations.

## 2. Related Work and Novel Contributions of QURSED Editor

The QURSED Editor relates to four wide classes of tools, coming from both academia and industry:

*Web-based Form and Report Generators* create web-based interfaces that access relational databases. Popular examples are Macromedia Dreamweaver UltraDev, ColdFusion, and Microsoft Visual InterDev. These tools are excellent when flat uniform relational tables need to be displayed. However, the development of form and report pages that query and display semistructured data requires substantial programming effort.

*Visual Querying Interfaces* are applications that allow the exploration of the schema and/or content of the underlying database and the formulation of queries. Typical examples are the Query-By-Example (QBE) [25] interface and Microsoft's Query Builder, which target the querying of relational databases. Recent visual front-ends such as EquiX [8], BBQ [18], VQBD [7], and PESTO [5] target the querying of XML and object-oriented databases. These systems provide an excellent visual paradigm for the formulation of fairly complex queries. Note though that they and the Editor have very different goals: The goal of the former is the development of a query or a query template by a database programmer, who is familiar with database models and languages. The goal of the latter is the construction from an average web developer of a form that represents and can generate a large number of possible queries.

*Schema Mapping Tools* are graphical user interfaces that declaratively transform data between XML Schemas in the context of integration applications. IBM's Clio [20], Microsoft's BizTalk Mapper, TIBCO's XML Transform, and Enosys's Query Builder [10] are representative examples. QURSED's Editor adopts part of the functionality provided by the schema mapping tools for a different purpose: it creates query/visual associations that map form controls on the XHTML query form page to parameters of selection predicates, in order to generate queries that filter the data. It also creates a transformation between a single XML Schema and an XHTML template report page in order to construct the report pages.

*Data-Intensive Web Site and Application Generators*. Autoweb [15], Araneus [3] and Strudel [13] are great examples of the ongoing research on how to design and develop web sites heavily dependent on database content by decoupling the query aspects of web development from the presentation ones. They all offer a data model, a navigation model and a presentation model. An extensive discussion of this class of systems can be found in [14].

## 2.1 Contributions

*Forms and Reports for Semistructured Data.* The QURSED Editor generates form and report specifications that target the needs of interacting with and presenting semistructured data. Multiple features contribute to addressing these needs:

1. QSS and QFRs support the generation of queries that handle the structural variance and irregularities of the source data by employing appropriate forms of disjunction. For example, consider a sensor query form that allows the end-user to check whether the sensor fits within an envelope with length *X* and width *Y*, where *X* and *Y* are end-user-provided parameters. The corresponding query has to take into consideration whether the sensor is cylindrical or rectangular, since *X* and *Y* have to be compared against a different set of dimension attributes in each case.
2. On the report side, data can be automatically nested according to the nesting proposed by the source schema or can be made to fit XHTML tables that have variance in their structure and different nesting patterns. The Editor supports both ways of producing nested data in report pages. Structural variance on the report page is tackled by producing heterogeneous rows/tuples in the resulting XHTML tables.

*Loose Coupling of Query and Visual Aspects.* The QURSED Editor separates the logical aspects of query forms and reports generation, i.e., the query form capabilities, from the presentation aspects, hence making it easier to develop and maintain the resulting form and report pages. The visual component of the forms can be prepared with any XHTML editor. Then the developer can focus on the logical aspects of the forms and reports: Which are the condition fragments? How should the report be nested? etc. The coupling between the logical and the visual part is loose, simple, and easy to build: The query parameters are associated with XHTML form controls, the condition fragments are associated with sets of XHTML form controls, and the *grouped* elements (see Section 3) of the result tree are associated with the nested tables of the report.

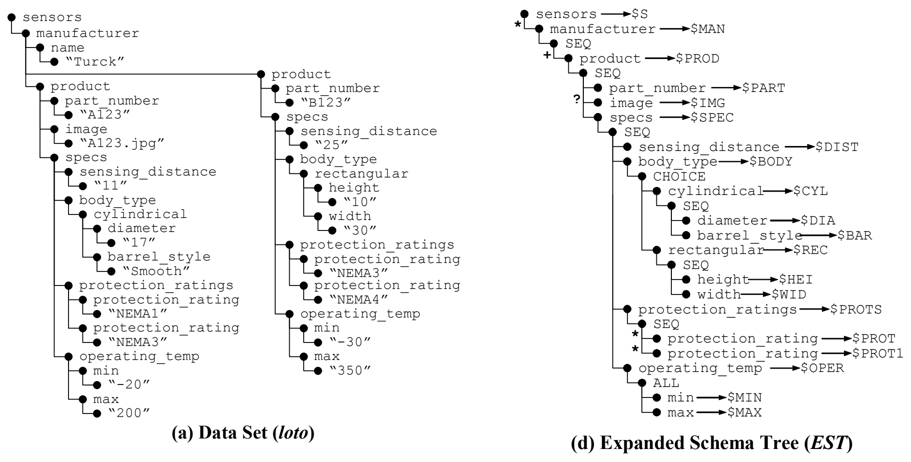


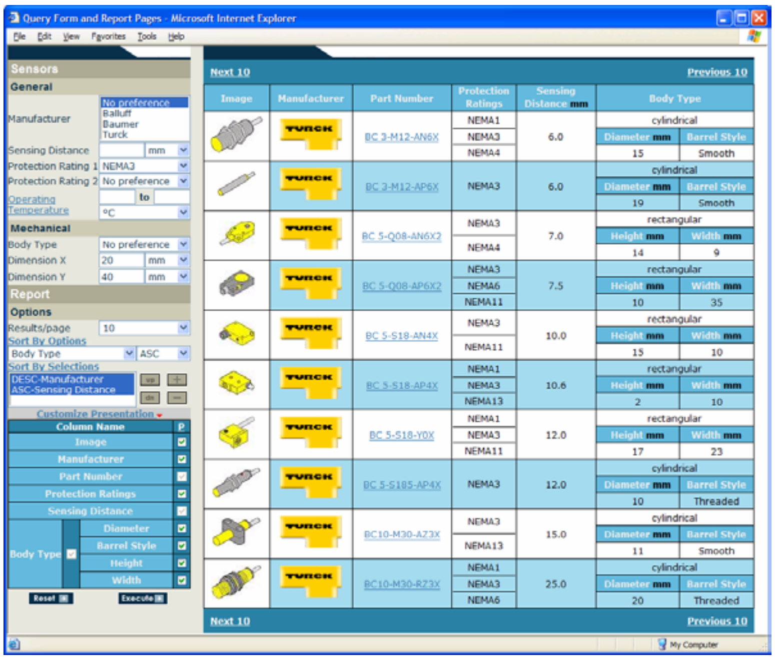
**Figure 2** Example Data Set and Expanded Schema Tree

## 3. The QURSED System

This section describes an example XML Schema, the corresponding *Expanded Schema Tree* and the data model of QURSED, and introduces as the running example a QURSED-generated web interface. It also describes the end-user experience with that interface and concludes by briefly presenting TQL, the internal query language used by QURSED, and QSS, the specification produced by the Editor.

### 3.1 Data Model, XML Schema and Expanded Schema Tree

QURSED models XML data as labeled ordered tree objects (*loto*s), such as the sample data set shown in Figure 2a that describes two proximity sensor products. Each internal node of the labeled ordered tree represents an XML element and is labeled with the element's tag name. The list of children of a node represents the sequence of elements that make up the content of the element. A leaf node holds the string value of its parent node. If *n* is a node of a *loto*, we denote as *tree*(*n*) the subtree rooted at *n*. Based on the XML Schema that describes the structure of the sample data set (not shown), the Editor constructs the corresponding Expanded Schema Tree (*EST*) that serves as the basis for building the query set specification. Figure 2b shows the representation of the *EST* by the Editor. Note that the root node of an *EST* is a non-repeatable element node. Also note that an *EST* can include multiple copies of the same element node, to allow the developer to create "aliases" of element nodes, each with unique element variables.


Query Form and Report Pages - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Reset
Execute
My Computer

Sensors

General

| | |
|---|---|
| Manufacturer | No preference / Balluff / Baumer / Turck |
| Sensing Distance | mm |
| Protection Rating 1 | NEMA3 |
| Protection Rating 2 | No preference |
| Operating Temperature | to °C |

Mechanical

| | |
|---|---|
| Body Type | No preference |
| Dimension X | 20 mm |
| Dimension Y | 40 mm |

Report

Options

| | |
|---|---|
| Results/page | 10 |
| Sort By Options | Body Type ASC |
| Sort By Selections | DESC-Manufacturer<br>ASC-Sensing Distance |

Customize Presentation

| Column Name | | | P |
|---|---|---|---|
| Image | | | ✓ |
| Manufacturer | | | ✓ |
| Part Number | | | ✓ |
| Protection Ratings | | | ✓ |
| Sensing Distance | | | ✓ |
| Body Type | ✓ | Diameter | ✓ |
| | | Barrel Style | ✓ |
| | | Height | ✓ |
| | | Width | ✓ |

Next 10 Previous 10

| Image | Manufacturer | Part Number | Protection Ratings | Sensing Distance mm | Body Type |
|---|---|---|---|---|---|
| | TURCK | BC 3-M12-AN6X | NEMA1<br>NEMA3<br>NEMA4 | 6.0 | cylindrical<br>Diameter mm: 15<br>Barrel Style: Smooth |
| | TURCK | BC 3-M12-AP6X | NEMA3 | 6.0 | cylindrical<br>Diameter mm: 19<br>Barrel Style: Smooth |
| | TURCK | BC 5-Q08-AN6X2 | NEMA3<br>NEMA4 | 7.0 | rectangular<br>Height mm: 14<br>Width mm: 9 |
| | TURCK | BC 5-Q08-AP6X2 | NEMA3<br>NEMA6<br>NEMA11 | 7.5 | rectangular<br>Height mm: 10<br>Width mm: 35 |
| | TURCK | BC 5-S18-AN4X | NEMA3<br>NEMA11 | 10.0 | rectangular<br>Height mm: 15<br>Width mm: 10 |
| | TURCK | BC 5-S18-AP4X | NEMA1<br>NEMA3<br>NEMA13 | 10.6 | rectangular<br>Height mm: 2<br>Width mm: 10 |
| | TURCK | BC 5-S18-Y0X | NEMA1<br>NEMA3<br>NEMA11 | 12.0 | rectangular<br>Height mm: 17<br>Width mm: 23 |
| | TURCK | BC 5-S185-AP4X | NEMA3 | 12.0 | cylindrical<br>Diameter mm: 10<br>Barrel Style: Threaded |
| | TURCK | BC10-M30-AZ3X | NEMA3<br>NEMA13 | 15.0 | cylindrical<br>Diameter mm: 11<br>Barrel Style: Smooth |
| | TURCK | BC10-M30-RZ3X | NEMA1<br>NEMA3<br>NEMA6 | 25.0 | cylindrical<br>Diameter mm: 20<br>Barrel Style: Threaded |

Next 10 Previous 10

**Figure 3** Example *QFR* Interface

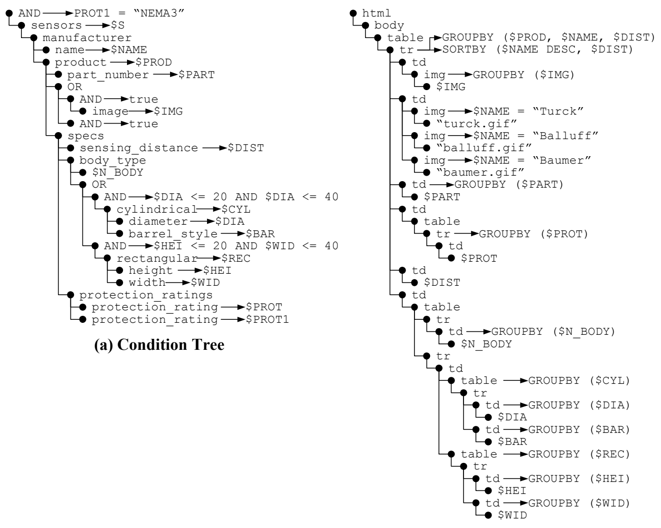
## 3.2 Example *QFR* and End-User Experience

Using QURSED, a developer can easily generate a web interface like the one shown in Figure 3 that queries and reports proximity sensor products. This interface will be the running example and illustrates the basic points of the functionality and the experience that QURSED delivers to the end-user of the interface.

The browser window displays a query form page and a report page. On the query form page form controls are displayed for the end-user to select or enter desired values of sensors' attributes and customize the report page. For example, the user has placed the equality condition "NEMA3" on "Protection Rating 1". After the end-user submits the form, she receives the report of Figure 3. The results depict the information of `product` elements: the developer had decided earlier that products should be returned. By default, QURSED organizes the presentation of the qualifying XML elements in a way that corresponds to the nesting suggested by their XML Schema. Notice, for example, that each product display has nested tables for `rectangular` and `cylindrical` values. Also notice that instead of the text of the manufacturer's name, a corresponding image (logo) is presented. Section 4 elaborates on the visual steps the developer follows on the Editor interface to deliver query form and report interfaces, like the one shown in Figure 3, using QURSED. The next section illustrates the query model and specification language of QURSED.

## 3.3 Tree Query Language (TQL) and Query Set Specification (QSS)

User interaction with the query form page results in the generation of TQL queries.



**(a) Condition Tree**

**(b) Result Tree**

**Figure 4** TQL Query Corresponding to Figure 3

A TQL query *q* consists of a *condition tree* and a *result tree*. An example of a TQL query is shown in Figure 4, and corresponds to the TQL query generated by the end-user's interaction with the query form page of Figure 3. Variables are denoted by the $ symbol. Note that conditions, in the form of Boolean expressions, are placed on diameter of cylindrical sensors and on height and width of rectangular sensors. The condition tree generates tuples of qualified *bindings* of variables and corresponds intuitively to the `WHERE` part of XML query languages such as XML-QL [9], and LOREL [21], and to the `FOR` and `WHERE` clauses of a `FLWOR` expression of the up-coming XQuery standard [6]. The result tree corresponds to the `CONSTRUCT` clause of XML-QL, the `SELECT` clause of LOREL, and the `RETURN` clause of a `FLWOR` expression of XQuery. A result tree specifies how to build new XML elements using the bindings provided by the condition tree. In particular, it specifies groupings of variables via *group-by* lists, and ordering via *sort-by* lists. Group-by and sort-by lists are the TQL means of performing grouping and sorting. Figure 4b shows the result tree for the example of Figure 3. Note that the rows of the XHTML tables that contain the static column names are omitted from the result tree for presentation clarity.

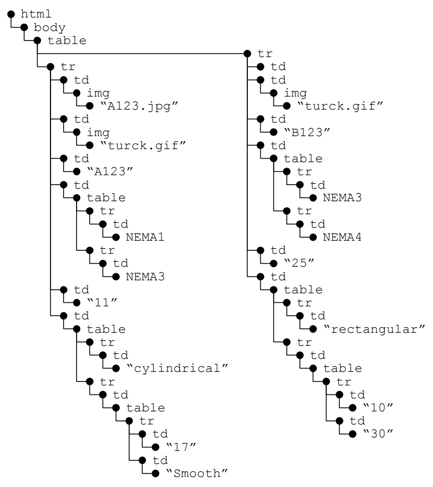


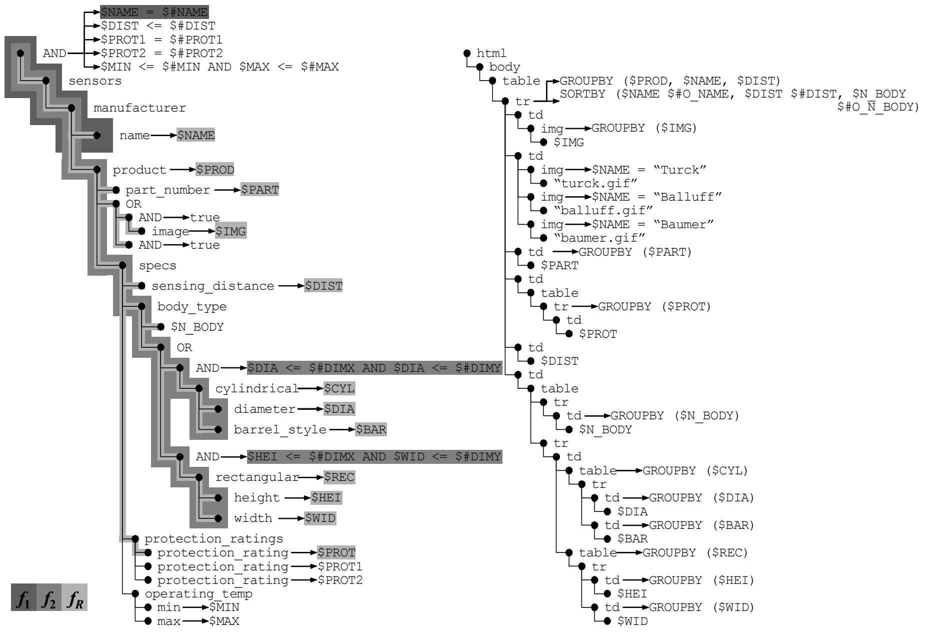
**Figure 5** Resulting *loto* for Query of Figure 4 run on the data of the source *loto* of Figure 2a

Figure 5 shows the resulting *loto* of the TQL query of Figure 4 run on the data of the source *loto* of Figure 2a. The TQL query generated by a query form page is a member of the set of queries encoded in the query set specification of the *QFR*. The QURSED system uses the TQL queries internally, but issues queries in XQuery [6] by translating TQL queries to equivalent XQuery statements.

QURSED uses query set specifications to succinctly encode in *QFR*s large numbers of possible queries, as the query set specification can describe a number of queries that is exponential in the size of the specification. The developer uses the Editor to visually create a query set specification, like the one in Figure 6. We briefly present the query set specification, as it is the underpinning of *QFR*s and the visual interfaces and interactions performed using the Editor as described in Section 4.

A query set specification *QSS* is a triple <*CTG*, *RTG*, *F*>, where:

- *CTG*, the *condition tree generator*, is a condition tree with multiple Boolean expressions possibly labeling each AND node and with parameters, as well as literal values, as operands of their predicates. Parameters are denoted by the $# symbol.
- *RTG*, the *result tree generator*, is simply a result tree with parameters, as well as values, as operands of the Boolean expression possibly labeling each node.
- *F* is a non-empty set of *condition fragments*. A condition fragment *f* is a subtree of the *CTG*, rooted at the root node of *CTG*, where each AND node is labeled with exactly one Boolean expression. *F* always contains a special condition fragment $f_R$, called *result fragment*, that includes all the element nodes whose variables appear in *RTG*, all its AND nodes are labeled with the Boolean value *true*, and has no parameters. The result fragment intuitively guarantees the "safety" of the result tree.



**(a) Condition Tree Generator** **(b) Result Tree Generator**

**Figure 6** Query Set Specification

For example, the query set specification of Figure 6 encodes, among others, the TQL query of Figure 4. The *CTG* in Figure 6a corresponds partially to the set *F* of condition fragments defined for the query form page of Figure 3. Three condition fragments are indicated with different shades of gray:

1. condition fragment $f_1$ is defined by the dark grey subtree and the Boolean expression on the root AND node of the *CTG* that applies a condition to the `name` element node;
2. condition fragment $f_2$ is defined by the medium gray subtree and the Boolean expressions that apply conditions to the dimensions of cylindrical and rectangular sensors ; and
3. condition fragment $f_R$ (the *result fragment*) is defined by the light grey subtree that includes all the element nodes whose variables appear in *RTG* in Figure 6b, and imposes no Boolean conditions.

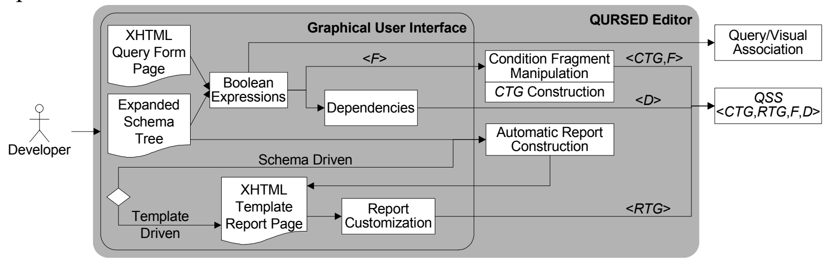


**Figure 7** QURSED Editor Architecture

## 4. QURSED Editor

The QURSED Editor is the tool the developer uses to build *QFR*s. Figure 7 shows the Editor's architecture, how the developer interacts with the graphical user interface, and how the Editor interprets these visual actions in order to construct the *QSS* and the query/visual association of a *QFR*.

The developer builds a condition tree generator by constructing a set of Boolean expressions based on the input XML Schema, in the form of an *EST*, and the input XHTML query form page that are displayed to her. Internally, the Editor interprets the set of Boolean expressions as both the set of condition fragments of the *QSS* and the query/visual specification. The Editor constructs the *CTG* by building each condition fragment *f*, as if *f* was the only fragment of the condition tree generator, and then merging *f* with the as-yet constructed *CTG*. A key step in the process is that the Editor checks if *f* is meaningful by considering the presence of CHOICE elements in the *EST* and, if necessary, manipulates *f* by heuristically introducing structural disjunction operators (OR nodes). This process is described in Section 4.1.

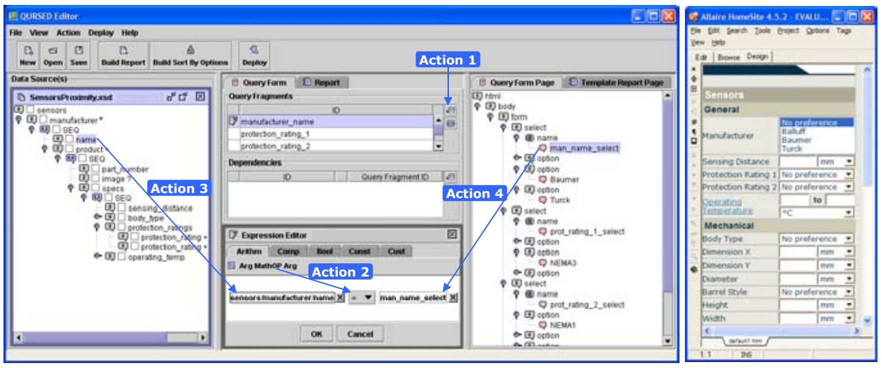
For the construction of the result tree generation, the developer has two options. Either an XTMHL template report page is automatically constructed based on the *EST* (schema-driven), or one is provided as an input (template-driven). Either way, the Editor constructs internally an *RTG* that becomes part of the *QSS*. Schema-driven result tree generation is described in Section 4.2.

A key benefit of the Editor is that it enables the easy generation of semistructured queries with OR nodes by considering the presence of CHOICE elements in the *EST*. The following subsections describe the visual actions and their translation to corresponding parts of the query set specification, using the *QSS* of Figure 6 and the *QFR* of Figure 3 as an example.

### 4.1 Building Condition Tree Generators

Figure 8a demonstrates how the developer uses the Editor to define the condition fragment $f_1$ of Figure 6a. The main window of the Editor presents the sample *EST* of Section 3.1 on the left panel, and the query form page on the right panel. The query form page is displayed as an XHTML tree that contains a form and a set of form controls, i.e., `select` and `input` element nodes [22]. The XHTML tree corresponds to the page shown on Figure 8b rendered in the Macromedia HomeSite WYSIWYG XHTML editor.

Based on this setting, the developer defines the condition fragment $f_1$ of Figure 6a that imposes an equality condition on the manufacturer's name by performing the four actions indicated by the arrows on Figure 8a. She starts by clicking on the "New Condition Fragment" button (Action 1 of Figure 8a) and providing a unique `ID`, which is `manufacturer_name` in this case. The middle panel lists the condition fragments defined so far, and the expression editor at the bottom allows their definition, inspection and revision. Then, the developer builds a Boolean expression in the expression editor, by drag 'n' dropping the equality predicate (Action 2) and setting its left operand to be the element node `name` (Action 3). The full path name of the node appears in the left operand box and is also indicated by the highlighting of the `name` element node on the left panel. As a final step, the developer binds the right operand of the equality predicate to the `select` XHTML form control named `man_name_select` (Action 4) thus establishing a query/visual association and defining the visual fragment that includes the "Manufacturer" form control shown in Figure 8b. Internally, the Editor creates the parameter `$#NAME`, associated with the "Manufacturer" form control Figure 8b, and sets it as the right operand of the Boolean expression, as Figure 6a shows.



**(a) QURSED Editor** **(b) WYSIWYG HTML Editor**

**Figure 8** Building a Condition Fragment

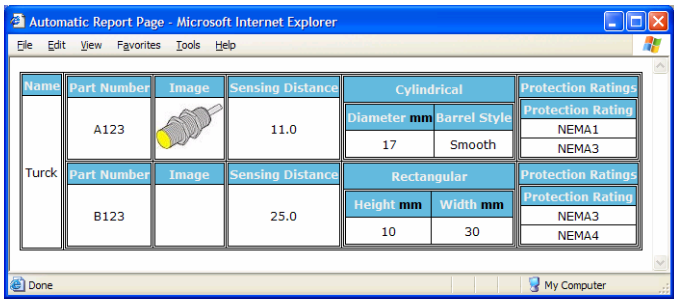
In order to build more complex condition fragments, Actions 2, 3 and 4 can be repeated multiple times, thus introducing multiple variable and parameters and including more than one XHTML form controls in the corresponding visual fragment. Note that, even though the visual actions introduce variables and parameters in the condition fragment, the developer does not need to be aware of them. In effect, variables correspond to path names and parameters to XHTML form control names. The Editor interprets the Boolean expression as a condition fragment that contains all paths of the expression.

**Automatic Introduction of Structural Disjunction.** The semistructuredness of the schema (CHOICE nodes and optional elements) may render the Boolean expression attached to a node meaningless and unsatisfiable. The Editor automatically, and by employing a heuristic, manipulates a condition fragment *f* by introducing structural disjunction operators (OR nodes) that render *f* meaningful. For example, consider the query form page of Figure 8b, where the end-user has the option to input two dimensions X and Y that define an envelope for the sensors, without specifying a particular body type. Sensors can be either cylindrical or rectangular. The developer's intention is to specify that either the diameter is less than dimensions X and Y, or the height is less than dimension X and the width less than Y. The developer constructs the following Boolean expression:

```
($DIA <= $#DIMX ∧ $DIA <= $#DIMY) ∨
 ($HEI <= $#DIMX ∧ $WID <= $#DIMY)
```

The `$DIA`, `$HEI` and `$WID` variables label the `diameter`, `height` and `width` elements of the *EST*. The `$#DIMX` and `$#DIMY` parameters are associated with the "Dimension X" and "Dimension Y" form controls.

However, the query where the above Boolean expression is interpreted as a condition fragment consisting of the paths to `diameter`, `height` and `width` elements is unsatisfiable, since no sensor has all of them. The Editor captures the original intention by automatically manipulating the $\vee$ Boolean connective and treating it as an OR node of TQL, as the condition fragment $f_2$ in Figure 6a indicates. The OR node corresponds to the CHOICE node in the *EST* of Figure 2c. Two AND nodes are also introduced and are labeled with the conjunctions in the initial Boolean expression: `($DIA <= $#DIMX ∧ $DIA <= $#DIMY)` and `($HEI <= $#DIMX ∧ $WID <= $#DIMY)`.



| Name | Part Number | Image | Sensing Distance | Cylindrical | | Protection Ratings |
|---|---|---|---|---|---|---|
| | | | | Diameter mm | Barrel Style | Protection Rating |
| Turck | A123 | | 11.0 | 17 | Smooth | NEMA1 |
| | | | | | | NEMA3 |
| | Part Number | Image | Sensing Distance | Rectangular | | Protection Ratings |
| | | | | Height mm | Width mm | Protection Rating |
| | B123 | | 25.0 | 10 | 30 | NEMA3 |
| | | | | | | NEMA4 |

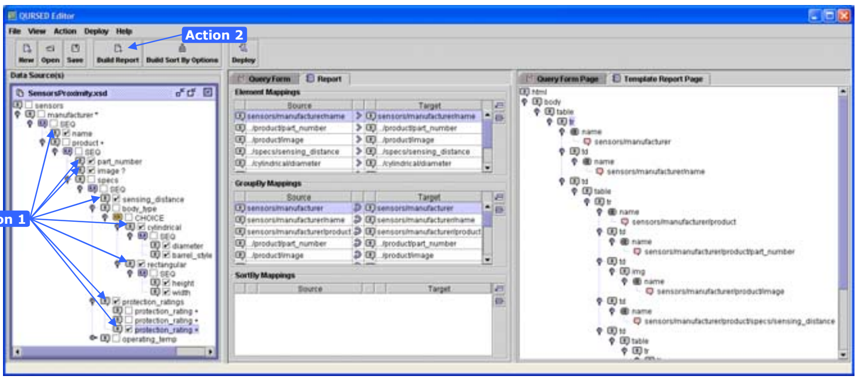
**Figure 9** Schema-Driven Constructed Report Page

The Editor creates a condition tree generator by appropriately merging the condition fragments. The merging algorithm operates incrementally by merging each condition fragment *f* with the condition tree generator already constructed from the previous condition fragments. The main step of the algorithm manipulates *f* by employing a heuristic, such that *f* produces meaningful satisfiable queries given the Boolean expression *b*. In particular, the algorithm introduces structural disjunction operators to *f* by replacing Boolean connectives $\vee$ in *b* with OR nodes, as illustrated in the example above. The manipulation is driven by the CHOICE nodes and optional elements (either ? or * occurrence constraint).

The merging algorithm often creates redundancies in the CTG. As shown in [2], efficiency of tree pattern queries depends on the size of the pattern, so it is essential to identify and eliminate redundant nodes. The Editor eliminates redundancies on the merged *CTG* in order to improve the performance of the generated TQL queries by using a set of rewriting rules. The rules preserve the boundaries of condition fragments as element nodes are being eliminated. The merging algorithm and the rewriting rules are described in [19].

## 4.2 Building Result Tree Generators

The Editor provides two options for the developer to build the result tree generator *RTG* component of a query set specification, each one associated with a set of corresponding actions. For the first (and simpler) option, called *schema-driven*, the developer only specifies which element nodes of the *EST* she wants to present on the report page. Then, the Editor automatically builds a result tree generator that creates report pages presenting the source data in the form of XHTML tables that are nested according to the nesting of the *EST*.



Element Mappings

| Source | Target |
|---|---|
| sensors/manufacturer/name | sensors/manufacturer/name |
| ../product/part_number | ../product/part_number |
| ../product/image | ../product/image |
| ../specs/sensing_distance | ../specs/sensing_distance |
| ../cylindrical/diameter | ../cylindrical/diameter |

GroupBy Mappings

| Source | Target |
|---|---|
| sensors/manufacturer | sensors/manufacturer |
| sensors/manufacturer/name | sensors/manufacturer/name |
| sensors/manufacturer/product | sensors/manufacturer/product |
| ../product/part_number | ../product/part_number |
| ../product/image | ../product/image |

SortBy Mappings

| Source | Target |
|---|---|

**Figure 10** Selecting Elements Nodes and Constructing Template Report Page

If the developer wants to structure the report page in a different way, the Editor provides a second option, called *template-driven*, where the developer provides as input a template report page to guide the result tree generator construction. Template-driven construction of an RTG is presented in [19].

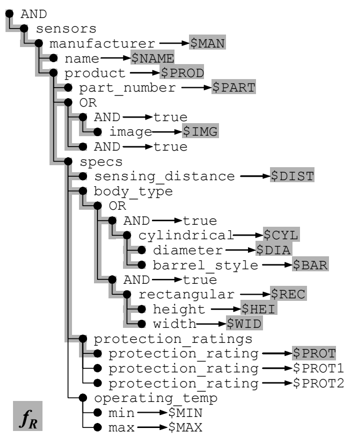
**Schema-Driven Construction of Result-Tree Generator.** The developer can automatically build a result tree generator based on the nesting of the *EST*. For example, Figure 9 shows a report page created from the result tree generator for the data set and the *EST* of Figure 2. The creation of the result tree generator and the template report page is accomplished by performing the two actions that are indicated by the numbered arrows on the Editor's window of Figure 10.

First, the developer uses the checkboxes that appear next to the element nodes of the *EST* to select the ones she wants to present on the report page (Action 1 of Figure 10). This action sets the *report* property of the selected element nodes in the *EST* to *true* and constructs the result fragment $f_R$ indicated in the condition tree generator of Figure 11a. The variables that will be used in the result tree generator are also indicated.

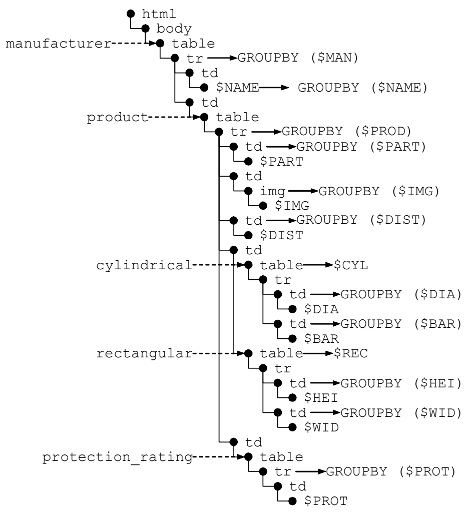
Then, the Editor automatically generates the template report page (Action 2) displayed on the right panel of Figure 10 as a tree of XHTML element nodes.

Figure 11c shows how a WYSIWYG XHTML editor renders the template report page. The element nodes selected in Action 1 are presented using XHTML `table` element nodes that are nested according to the nesting of the *EST*.
For illustration purposes, each `table` element node in Figure 11b is annotated with the *EST* element node it corresponds to, e.g., the "product" table is nested in the "manufacturer" table, as is the case in the *EST*. The table headers in Figure 11c are created from the name labels of the selected element nodes. In the tables, the Editor places the element variables of the element nodes selected in Action 1 as children of

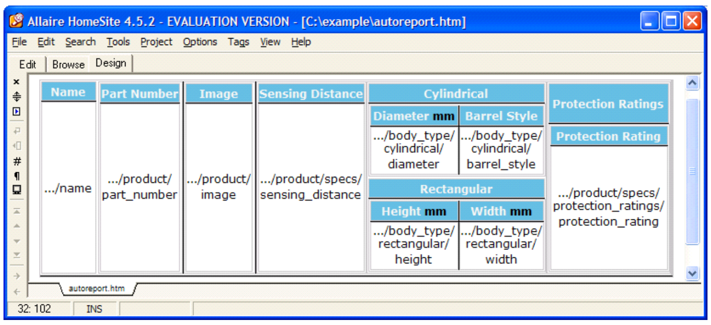
`td` (table data cell) element nodes. For example, in the result tree generator of Figure 11b the element variable `$NAME` appears as the child of the `td` element node of the "manufacturer" table.



(a) Condition Tree Generator



(b) Result Tree Generator



Allaire HomeSite 4.5.2 - EVALUATION VERSION - [C:\example\autoreport.htm]
File Edit Search Tools Project Options Tags View Help
Edit Browse Design
autoreport.htm
32: 102 INS

| Name | Part Number | Image | Sensing Distance | Cylindrical | | Protection Ratings |
|---|---|---|---|---|---|---|
| | | | | Diameter mm | Barrel Style | Protection Rating |
| .../name | .../product/ part_number | .../product/ image | .../product/specs/ sensing_distance | .../body_type/ cylindrical/ diameter | .../body_type/ cylindrical/ barrel_style | .../product/specs/ protection_ratings/ protection_rating |
| | | | | Rectangular | | |
| | | | | Height mm | Width mm | |
| | | | | .../body_type/ rectangular/ height | .../body_type/ rectangular/ width | |

(c) Template Report Page

**Figure 11** Automatically Generated Result Fragment, Result Tree Generator and Template Report Page

We discuss next how repeatable element nodes are managed. The complete algorithm, called *AutoReport*, for constructing the result fragment and the result tree generator, is presented in [19], and handles every structure of *EST*.

The Editor handles repeatable element nodes in the *EST* by automatically generating corresponding `table` elements and group-by lists in the result tree generator. For example, the path from the root of the *EST* to the `name` element node that is selected in Action 1 contains the `manufacturer` repeatable element node, which results in the generation of the "manufacturer" `table` element node, shown in **Figure** 11b, and the group-by list of its `tr` (table row) child element node. This group-by list will generate one table row for each binding of the `$MAN` element variable.

## References

[1] S. Abiteboul, P. Buneman, D. Suciu: *Data on the Web*, Morgan Kaufman, CA (2000)

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, D. Srivastava: Minimization of Tree Pattern Queries, *Proceedings* ACM SIGMOD Conference, (2001)

[3] P. Atzeni, G. Mecca, P. Merialdo: To Weave the Web, *Proceedings 23rd VLDB Conference*, (1997)

[4] P. Bernstein et al.: The Asilomar report on database research, *ACM SIGMOD Record* 27(4) (1998)

[5] M. Carey, L. Haas, V. Maganty, J. Williams: PESTO: An Integrated Query/Browser for Object Databases, *Proceedings 22nd VLDB Conference* (1996)

[6] D. Chamberlin et al.: XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/

[7] S. Chawathe, T. Baby, J Yeo: VQBD: Exploring Semistructured Data (demonstration description), *Proceedings* ACM SIGMOD Conference (2001)

[8] S. Cohen, Y. Kanza, Y. Kogan, W. Nutt, Y. Sagiv, A. Serebrenik: EquiX – Easy Querying in XML Databases, *Proceedings ACM WebDB Workshop*, (1999)

[9] Deutsch et al.: XML-QL: A Query Language for XML, W3C note (1998) http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[10] Y. Papakonstantinou, V. Vassalos: The Enosys Data Integration Platform: Lessons from the Trenches, *Proceedings 10th CIKM Conference*, (2001)

[11] M. Fernández et al.: XQuery 1.0 and XPath 2.0 Data Model, http://www.w3.org/TR/query-datamodel/

[12] M. Fernández, A. Morishima, D. Suciu: Efficient Evaluation of XML Middle-ware Queries, *Proceedings ACM SIGMOD Conference* (2001)

[13] M. Fernández, D. Suciu and I. Tatarinov: Declarative Specification of Data-intensive Web sites, *Proceedings Workshop on Domain Specific Languages (DSL),* (1999)

[14] P. Fraternali: Tools and Approaches for Data Intensive Web Application Development: a Survey, *ACM Computing Surveys* 31(3) (1999)

[15] P. Fraternali, P. Paolini: Model-Driven Development of Web Applications: the Autoweb System, *ACM Transactions on Office Information Systems* 18 (4) (2000)

[16] Levy, A. Rajaraman, J. D. Ullman: Answering Queries Using Limited External Processors, *Proceedings* PODS Symposium, (1996)

[17] Ludäscher, Y. Papakonstantinou, P. Velikhov: Navigation-Driven Evaluation of Virtual Mediated Views, *Proceedings EDBT Conference*, (2000)

[18] K. Munroe, Y. Papakonstantinou: BBQ: A Visual Interface for Browsing and Querying XML, *Proceedings 5th Visual Database Systems Conference* (2000)

[19] Y. Papakonstantinou, M. Petropoulos, V. Vassalos: Graphical Query Interfaces for Semistructured Data: The QURSED system, submitted for publication

[20] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, R. Fagin: Translating Web Data, *Proceedings 28th VLDB Conference*, (2002)

[21] D. Quass et al.: Querying Semistructured Heterogeneous Information, *Proceedings 4th DOOD Conference*, (1995)

[22] D. Raggett, A. Le Hors, I. Jacobs: HTML 4.01 Specification, http://www.w3.org/TR/html4/

[23] H. Schöning, J. Wäsch: Tamino - an Internet Database System, *Proceedings EDBT Conference* (2000)

[24] V. Vassalos, Y. Papakonstantinou: Expressive Capabilities Description Languages and Query Rewriting Algorithms, *Journal of Logic Programming*, 43(1) (2000)

[25] M. Zloof: Query By Example, *Proceedings National Computer Conference*, AFIPS, Vol. 44 (1975)