Supporting Model-based Construction of Semantic-enabled Web Applications

Fuchs M., Niederée C., Hemmje M., Neuhold E.-J.
Fraunhofer Institute IPSI - 64293 Darmstadt, Germany
{fuchs, niederee, hemmje, neuhold}@ipsi.fhg.de

Abstract

Semantic annotation of Web content is in the core of the current Semantic Web Activity. The operationalization of the Semantic Web raises the challenge on how to systematically integrate semantic annotations into generated Web application pages.

In this paper, we present VizCo*, a tool for systematically integrating RDF based semantically enriched application domain models into the process of setting up dynamically generated Web application user interfaces and supporting their evolution. The form-based Web pages are annotated based on the underlying domain model.*

Combined with further mapping tools, VizCo *is used in our Web application development framework for coupling the domain model views and, indirectly, the underlying application data with other Web application components, especially with elements of the user interface. In contrast to a direct coupling with the application data an additional semantic layer is introduced. We follow a pragmatic approach that utilizes domain information extracted from application schemata and data as a starting point. Both the domain model and its coupling to the data source can be manually refined and extended. The couplings are dynamically translated into bi-directional data bindings at runtime.*

1 Introduction

The usefulness of semantic annotation of Web content has been impressively illustrated in [2]. The operationalization of the Semantic Web raises the question of where this semantic annotation should come from, since, with the vast amount of available content, a purely manual approach is not tractable. We believe that, in addition to the analysis and semi-automatic extraction of semantic information from structured and semi-structured content like Web Pages, text documents, etc., valuable starting points for semantic annotation can also be induced from structured information basis, i.e., data stored in databases. Considering the percentage of dynamic Web application pages generated from database content in today's Web shows the potential relevance of this source for semantic annotation.

This paper addresses the issue on how to smoothly integrate semantic annotation into Web application development enabling the inclusion of semantic annotations into dynamically generated Web application pages. The semantic annotation is based on the underlying domain model. The presented work can, thus, be considered as a contribution towards the operationalization of the Semantic Web.

In more detail, we present *VizCo*, a tool for visually defining application specific views and flexible application data couplings for RDF-based domain models. Combined with other components of our model-based Web application development framework, *VizCo* enables the visual definition of couplings between a domain model and other objects of Web application like user interface components. The domain model is an enriched view of the underlying application data, which is coupled with this data. The tool takes an RDF-based description of the domain model as an input and produces an RDF expression describing the coupling interactively defined. At Web application runtime such expression is automatically translated into a bi-directional binding with the underlying application data controlling the data displayed in the Web pages as well as the propagation of data modifications resulting from user interactions.

The design of *VizCo* is inspired by work in the area of visual support for query formulation [21, 1], where visual user interaction with a graph replaces the use of a textual query language prioritizing ease of use over full expressive power. In our tool the graphical paradigm is applied to a (simplified) RDF graph representation and produces domain model view definitions with an underlying coupling to the application data. The interactively created domain view definition expressions can be used by other components to couple Web application objects like user interface elements with the domain model and indirectly with a semantically enriched version of the underlying application data.

The rest of the paper is structured as follows: Section 2 discusses related work in the areas of visual query support, RDF query languages, and the model-based software development approach. Section 3 describes the conceptual ap-

proach underlying the *VizCo* tool including an overview of the coupling definition process, the conversion rules as well as the representation languages for domain model views and couplings. *VizCo* is part of our framework for the model-based development and evolution of semantic-enabled Web applications. An overview of this framework is given in section 4 before relevant issue of the prototypical implementation of *VizCo* are described. The paper concludes with a summary and with an outlook to future work.

2 Related Work

The visual definition of domain model views and couplings in *VizCo* is closely related to work in the area of visual support for query formulation, which has a long tradition going back to early approaches like QBE [21]. More recent approaches like [1, 4, 9] use graph-oriented representations of the data schema for facilitating query formulation as well as schema navigation and understanding. In addition, the G+ project [4], which is based on the Graphlog query language also uses graphical representations of instance data and of query results.

Most similar to the visual definition approach used in *VizCo* is the tool QBD* [1]. Like QBD*, *VizCo* uses a graph and an isomorphic language expression to represent the query which is interactively constructed by the user. The language expression is later translated into an expression of the target query language (SQL). We also follow a two level selection approach enabling the user to restrict the underlying schema (of the domain model) to a *schema of interest* before starting with the visual composition of the selection expression, which is especially useful when working with large schemata. In contrast to the QBD* approach, which is based on the Entity Relationship model [3], *VizCo* operates on an RDF Schema and is tailored to its property centric data model. Furthermore, the expressions constructed with *VizCo* are no queries but coupling expressions that are used at runtime to establish bi-directional bindings.

The domain selection and coupling expressions produced by *VizCo* are based on an RDF query language. Currently, several proposals for RDF query languages are under development [10, 12, 13, 19], but it is not yet clear how a possible standard in this area will look like. So, we have chosen one of the existing approaches as a basis, namely RDF Query developed at IBM [11], because queries themselves are formulated in XML and they are based on a clear declarative approach. Currently, we use the RDF query expressions as an exchange format for coupling expressions. There is no ”native” RDF query engine involved, since the query expressions are mapped to SQL queries.

VizCo is embedded within a model-based framework, following the MVC (Model-View-Controller) approach. Similar model-based frameworks are [17], [18], [7], [20].

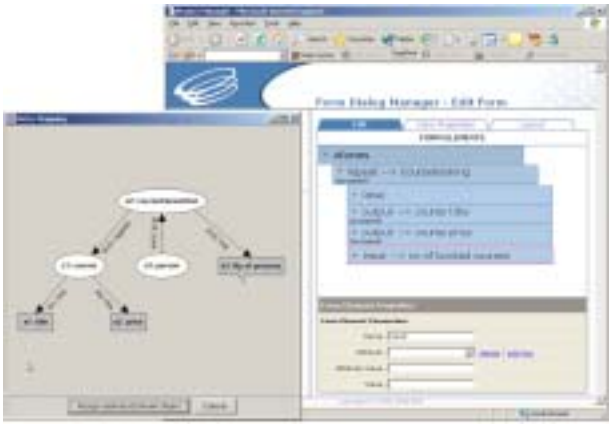

Figure 1. VizCo Mapping to user interface

Mobi-D [17] is a interactive environment where declarative models can be connected (see [16]). Mobi-D distinguishes two different kinds of models, the abstract and the concrete models. All combinations of mapping are possible between abstract and concrete. In *Mecano* [18], a model-based user interface development environment is introduced, which provides a tool for creating domain models. Based on this model, *high-level* dialogs (e.g. workflow and navigation structure of windows) as well as *low-level* dialogs (one step within the high-level dialog e.g. form input field, button) can be generated and later customized. In contrast to our approach the domain model has to be constructed manually and is not directly coupled to existing application data. *SUIMS*, another model-based user interface construction approach described in [7] is based on the so-called *ART Schemata* where a user interface model is composed from objects, actions, parameter, attributes and their types, pre-conditions, and post-conditions as well as the corresponding relations between them. These schemata can be instantiated with different parameters resulting in the creation of different types of user interfaces. To build up the schemata, a highly skilled programmer is needed.

3 The *VizCo* Coupling Definition Process

3.1 Example Usage Scenario

Coupling the application data with the domain model is, not a value by its own. It has to be considered as part of a larger scenario, where domain model objects are used in a Web application and the coupling of the domain model with the underlying application data assures consistent data management. We, therefore, consider an example usage scenario for motivating our approach: *VizCo* is integrated into the Form Dialog Manager (FDM), a central component of our Web application authoring tools suite. The FDM enables the composition of form-based Web appli-

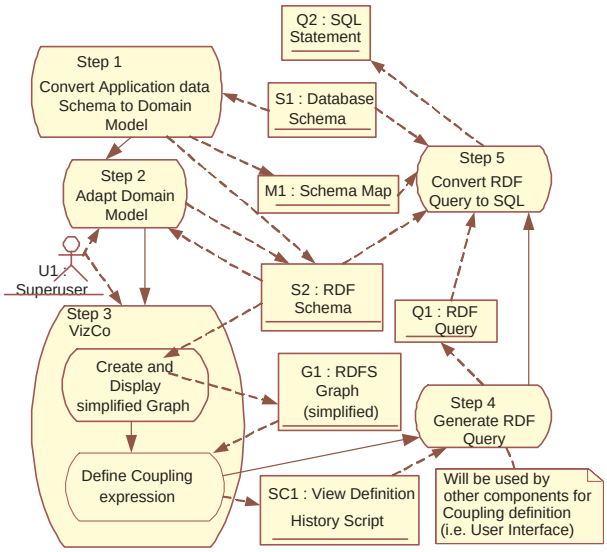

Figure 2. The"Coupling" process

cation user interfaces. It is based on XForms [5] the upcoming W3C standard for form-based Web pages, which follows the MVC approach by clearly separating layout aspects from data and control logic and is equipped with a more powerful client interaction model.

After defining a form field with the FDM like e.g. the input field "number of persons", *VizCo* is called (see figure1). *VizCo* exposes the domain model and enables the developer to interactively select the parts of the domain model that are to be coupled with the form field. An expression describing the composed view on the domain model is integrated into the Web page. At Web application runtime this expression is used (in combination with session information) to fill the form field with a value and to consistently write back the changes made in the form field by the user.

*VizCo*can also be easily integrated into other components, where a coupling with domain model objects is required, like e.g. for customizing a data analysis engine.

3.2 Overview of the Coupling Definition Process

It is the task of *VizCo* to enable the definition of domain model views that become part of coupling expressions. The UML activity diagram in figure 2 shows the most important steps of the *VizCo* coupling definition process as well as the object flow i.e. the input and output of each activity:

Step 1: A domain model is created by automatically converting the schema of the application data and into an RDF Schema that represents the domain model;

Step 2: This domain model can be refined and adapted by the user by selecting, renaming and (semantically) enriching the concepts and properties of the RDF Schema;

Step 3: A simplified RDF graph representation of this domain model of interest is used as a starting point for the definition of the domain view expression. This graph is displayed and the user interactively composes the domain view expression, which is represented by a graph and by a textual interaction history expression;

Step 4: Upon completion of the interaction, the history expression is converted into an RDF query expression that is used as a format for exchanging coupling expressions with other components of the framework;

Step 5: Steps 1-4 are referring to the design time of the system. At runtime the coupling defined at design time is translated into a binding to the application data. This means that query language expressions for accessing the respective data and its manipulation language expression for modifying the respective data after user interaction are generated. As a target language we use SQL;

The automatic mappings used in the process as well as the language used for representing the interaction history are described in the following sections.

3.3 Schema Conversion

Step 1 of the domain coupling definition process is the schema conversion, which generates an RDF Schema (representing the domain model) from the database schema of the underlying application data. For this purpose a mapping between the relational schema of the application data and the RDF Schema of the domain model is defined. A bijective mapping is required, since we need a coupling between the two models, that does not only support access, but also modification of application data via the domain model.

The mapping consists of two parts: An implicit part that defines a set of rules to be applied for performing the mapping and an explicit part, which stores information created during the generation and refinement of the RDF Schema and reuses them when the coupling is exploited at runtime.

For defining the mapping we need a function f_{smap} that takes an relational database schema as an input and creates an RDF schema as an output and the inverse function g_{smap}.

On a closer look, there are three types of entities in the relational schema that have to be taken into account for the mapping: The tables themselves, the columns of the tables and the relations between tables encoded into foreign key/primary key value pairs. Currently, we are not considering additional constraints (beyond primary key and referential integrity) in our mapping. For the definition of the mapping we assume that each of these entities is represented by a tuple (see definition D1): Each table is represented by its name and its primary key, each column, besides the ones representing foreign keys, is represented by its name, its type and the table it is associated with, and each relation is represented by the name of the table containing the foreign key, the name of the foreign key and the name of the table that the foreign key refers to (primary key table).

Definition D1:

$$table := \left\langle \begin{array}{c} <tableName> \\ <primaryKey> \end{array} \right\rangle,$$
$$column := \left\langle \begin{array}{c} <columnName> \\ <columnType> \\ <tableName> \end{array} \right\rangle,$$
$$relation := \left\langle \begin{array}{c} <foreignKeyTab> \\ <foreignKey> \\ <primaryKeyTab> \end{array} \right\rangle$$

The name of the table (<tableName>) is unique within the database and will be considered as an "identifier" for the database table. $primaryKey$ consists of one or more columns. The pair <columnName>.<tableName> is unique within the database and will be considered as an "identifier" for the column; <columnType> represents the datatype of the column. <foreignkeytab> and <primarykeytab> are the names of the tables where the <foreignkey> and the referred <primarykey>, respectively, are located.

Using this representation the relational schema of the application data S_{DB} can be represented by the set M_{tab} of all tables, the set M_{col} of all columns, and the set M_{rel} of all relations. This results in the following definition for the domain of our mapping function $fsmap$:

Definition D2: $D(f_{smap}) := S_{DB} = (M_{tab}, M_{col}, M_{rel})$.

In the RDF schema there are two type of entities: RDF classes and RDF properties. Furthermore, we distinguish RDF properties that use literals as a range from those who have another class as a range. The second group of properties represents a relationship between two RDF classes. We call it a relationship property (*relproperty*). Each of these entities is represented by a tuple (see definition D3).

Definition D3: $$class := \left\langle \begin{array}{c} <URI> \\ <label> \end{array} \right\rangle, prop := \left\langle \begin{array}{c} <URI> \\ <label> \\ <domain> \\ <range> \end{array} \right\rangle,$$

$$relprop := \left\langle \begin{array}{c} <URI> \\ <label> \\ <domain> \\ <range> \end{array} \right\rangle$$

Each of the RDF Schema elements contains a URI (Uniform Resource Identifier) as global identifier and a label as human readable identifier. Properties contain the reference to the domain and range of the property as implied by the property-centric RDF data model.

f_{smap} can now be summarized as follows: Tables are mapped to RDF classes, columns in the tables are mapped to properties and the relations are mapped to RDF relation properties. In more detail, three functions f_{tab}, f_{col}, and f_{rel} are defined.

The function for mapping a table to an RDF class mainly consist of the application of two naming functions ID_{class} and $ID_{classLabel}$ that encode conventions for the creation of the RDF class URI and the class label from the table name.

Definition D4:

$$f_{tab}(table) = f_{tab}\left(\left\langle \begin{array}{c} <tableName> \\ <primaryKey> \end{array} \right\rangle\right) := \left\langle \begin{array}{c} <ID_{class}(tableName)> \\ <ID_{classLabel}(tableName)> \end{array} \right\rangle$$

Mapping Type	Database Schema Element	RDF Schema Element
table <->class	course	http://www.example.de/DM#course
column <-> property	course.title	http://www.example.de/DM#course_title
relation <-> relproperty	course.domainIDREF	http://www.example.de/DM#courseTOdomainREL
table <->class	domain	http://www.example.de/DM#domain
column <-> property	domain.name	http://www.example.de/DM#domain_name
table <->class	person	http://www.example.de/DM#person
column <-> property	person.name	http://www.example.de/DM#person_name
table <->class	student	http://www.example.de/DM#course
column <-> property	student.mn	http://www.example.de/DM#course
relation <-> relproperty	student.personIDREF	http://www.example.de/DM#course

Table 1. Stored name mappings (example)

The generated URIs and labels of the classes are stored together with the associated table names as part of the mapping information (explicit part of the mapping). This is necessary, because the user may manually adapt the labels and URIs of RDF classes in the domain model. Table 1 shows an example of a name mapping table.

A separate RDF property has to be created for each table column. So, for each column of each table the function f_{col} creates the respective RDF property. This mapping is performed for all columns that are not part of a foreign key. In analogy to the class mapping the URI and the label of the RDF property are created by the functions ID_{prop} and $ID_{propLabel}$, respectively.

Definition D5: $$f_{col}(column) = f_{col}\left(\left\langle \begin{array}{c} <columnName> \\ <columnType> \\ <tableName> \end{array} \right\rangle\right) := \left\langle \begin{array}{c} <ID_{prop}(columnName)> \\ <ID_{propLabel}(columnName)> \\ <ID_{class}(tableName)> \\ <converType(columnType)> \end{array} \right\rangle$$

For each relation represented in the schema S_{DB} a relation property has to be created. This relation property takes the class associated with the table containing the foreign key as the domain of the property and the class associated with the table containing the primary key as range of the property. The naming function ID_{class} is used to create the URIs referring to the respective classes. The function convert type converts the application data types to the types of the domain model. Currently, we are mapping all types to the Literal class of RDF. In the next version typed literals based on the type system of XML Schema will be used.

Definition D6: $$f_{rel}(rel) = f_{rel}\left(\left\langle \begin{array}{c} <foreignKey> \\ <foreignKeyTab> \\ <primaryKeyTab> \end{array} \right\rangle\right) := \left\langle \begin{array}{c} <ID_{prop}(foreignKey)> \\ <ID_{propLabel}(foreignKey)> \\ <ID_{class}(foreignKeyTab)> \\ <ID_{class}(primaryKeyTab)> \end{array} \right\rangle$$

The application of the three functions to all the tables, columns and relations of the application schema, respectively, results in the RDF schema S_{RDF} that consists of the set M_{class} of RDF classes and the sets M_{prop} of RDF properties with an literal as range and $M_{relprop}$ of RDF properties with a class as range (see Definitions D7 and D8).

Definition D7:

$$M_{class} = \bigcup_{t \in M_{tab}} f_{tab}(t) \wedge M_{prop} = \bigcup_{c \in M_{col}} f_{col}(c) \wedge M_{relprop} = \bigcup_{r \in M_{rel}} f_{rel}(r)$$

Definition D8:

$$S_{RDF} = (M_{class}, M_{prop}, M_{relprop})$$

Summarizing all the definitions and considerations the mapping function f_{smap} is defined as follows:
Definition D9:

$$\vec{f}_{smap}(S_{DB}) = \vec{f}_{smap}\left(\left\langle \begin{smallmatrix} M_{tab} \\ M_{col} \\ M_{rel} \end{smallmatrix} \right\rangle\right) = \left\langle \begin{smallmatrix} f_{tab}(M_{tab}) \\ f_{col}(M_{col}) \\ f_{rel}(M_{rel}) \end{smallmatrix} \right\rangle = \left\langle \begin{smallmatrix} M_{class} \\ M_{prop} \\ M_{relprop} \end{smallmatrix} \right\rangle$$

Correspondingly, the inverse function g_{smap} (= f^{-1}_{smap}) for mapping elements of the created RDF schema back to the relational schema is defined. Note, that g_{smap} is tailored to the generated RDF schemata. It is not a general function to map an arbitrary RDF schema to a data base representation.

3.4 RDF Schema Element Management

In order to tailor the domain model to the "model of interest" (in analogy the "schema of interest" [1]), the framework supports RDF Schema element management activities in step 2 of the domain model view and coupling definition process. This includes the enrichment of RDF schema elements with the semantic annotation like the coupling of a class or property in the domain model with an ontology or a published XML vocabulary. Such enrichment is propagated by the framework to the Web application (user) interface and can, thus, be exploited by other applications. Furthermore, it is possible to restrict the domain model to a subschema to better focus the coupling activities in the Web application development process.

3.5 Visual Coupling Definition using *VizCo*

In step three of the Coupling definition process the domain model (an RDF graph) is represented by *VizCo* and the user interactively composes a second graph that represents the view definition. This section describes the simplified graph representation used to visualize the RDF Schema as well as the language that is used to represent the action history of the view graph composition.

3.5.1 Simplified RDF Graph Representation

RDF Schema (RDFS) is itself expressed in RDF and can thus be represented as a directed labeled graph or in a serialized form like e.g. RDF/XML. Since ease of use is one of the design goals of *VizCo*, a graph representation is used in this tool to visualize the domain model as well as the domain view underlying the coupling definition. This second graph is called *coupling definition graph*. In order to reduce graph size, we introduce two simplifications into the RDF graph representation used in *VizCo* resulting into a "Simplified RDF Graph (SRG)":

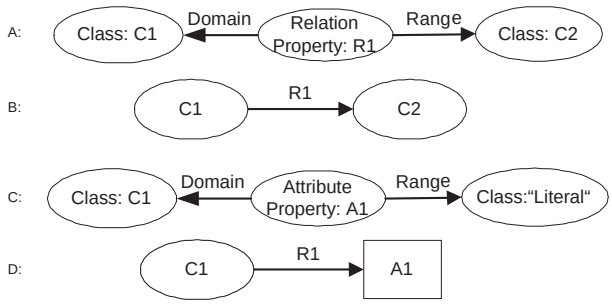

Figure 3. RDFS graph simplifications

1. In the graph representation of an RDF Schema the relation between two classes is represented as depicted in figure 3A: The relation itself is a first class object represented as a resource that is connected via the *domain* and the *range* property with the two classes. In SRG the two arrows and the resource representing the relation property is replaced by a single arrow as depicted in figure 3B pointing from the domain class to the range class. Note, that this simplification is only possible, if the relation property R1 is not involved in relationships to other resources.

2. Similarly, properties that relate a class to a literal are simplified in SRG as shown in figures 3C and 3D. The resource representing the property A1 and the resource representing the RDF class *Literal* are replaced by the single resource A1 for which the RDF graphs Literal symbol, the box, is used and an arrow pointing from the class to the literal. Again, this is only possible, if the property is not involved in other relationships.

An example coupling definition graph as it is visualized by *VizCo* is shown in figure 5 (see section 4).

3.5.2 The Action History Language

The user interactions with *VizCo* during domain view definition in step 3 of the coupling definition process are captured by expressions of an action history language. The syntax of this language is described by the BNF grammar depicted in figure 4. Starting from the empty graph the language describes the step-by-step construction of the coupling definition graph. For this purpose it contains actions to add elements to the coupling definition graph and to move the focus in the graph. Furthermore, it is also possible that a user withdraws elements from the coupling definition graph during a session. The top level rule of the grammar (defining $actionGraph$) reflects these three categories of actions:

Constructive actions: $addToGraph$ defines five ways for extending the graph. With $addClassToGraph\ (op1)$, $addPropertyToGraph\ (op2)$, and $addRelationToGraph\ (op3)$ the basic SRG elements can be added to the coupling graph.

```
actionGraph                 ::= addToGraph | moveFocus | withdrawFromGraph |emptyGraph

withdrawFromGraph           ::= withdrawClassFromGraph | withdrawPropertyFromGraph
                                | withdrawRelationFromGraph

addToGraph                  ::= addClassToGraph | addPropertyToGraph | addCouplingFlagToGraph
                                | addRelationToGraph | addConditionToGraph

moveFocus                   ::= moveFocusToClass | moveFocusToProperty

addClassToGraph             ::= 'addClassToGraph(' actionGraph ',' class ')'
addPropertyToGraph          ::= 'addPropertyToGraph(' actionGraph ',' property ')'
addRelationToGraph          ::= 'addRelationToGraph(' actionGraph ',' relation ')'
addConditionToGraph         ::= 'addConditionToGraph(' actionGraph ',' compareExp ',' property ')'
addCouplingFlagToGraph      ::= 'addCouplingFlagToGraph(' actionGraph ',' option ',' property ')'

withdrawClassFromGraph      ::= 'withdrawClassFromGraph(' actionGraph ',' class ')'
withdrawPropertyFromGraph   ::= 'withdrawPropertyFromGraph(' actionGraph ',' property ')'
withdrawRelationFromGraph   ::= 'withdrawRelationFromGraph(' actionGraph ',' relation ')'

moveFocusToClass            ::= 'moveFocusToClass(' actionGraph ',' class ')'
moveFocusToProperty         ::= 'moveFocusToProperty(' actionGraph ',' property ')'

class                       ::= elementName ',' elementAlias ',' elementURI
property                    ::= elementName ',' elementAlias ',' elementURI ',' domainClassAlias
relation                    ::= elementName ',' elementAlias ',' elementURI

emptyGraph                  ::= 'emptyGraph'
elementName                 ::= LITERAL
elementAlias                ::= LITERAL
elementURI                  ::= LITERAL
compareExp                  ::= LITERAL
domainClassAlias            ::= LITERAL
option                      ::= 'yes' | 'no'
```

Figure 4. BNF Grammar for the action history

The rule $addConditionToGraph$ $(op4)$ is contained within the grammar to define a condition for a domain view definition. Conditions are added to properties contained in the coupling definition graph. The condition contains a comparison expression $compareExp$. The substructure of this expression is not further analyzed by the grammar.

$addCouplingFlagToGraph$ refers to a property and determines, if this property becomes part of the dfined domain view. The $option$ rule defines the options "yes"$(op5)$ and "no"$(op6)$ for this purpose.

Destructive actions: $withdrawFromGraph$ withdraws either a class ($withdrawClassFromGraph$), a property ($withdrawPropertyFromGraph$), or a relation ($withdrawRelationFromGraph$) from the coupling definition graph. In order to have a connected graph, withdraw can only be applied on graph nodes, which are related at most with one other node (due to the *VizCo* rules, only the first selected class is a class without relations to another node). Automatically the relation, if existent, will also be withdrawn by applying the $withdrawRelationFromGraph$ rule. Arcs (relations) can only be removed from the graph, when this is not the last arc connecting.

Navigation actions: $moveFocus$ supports the rule-restricted control of navigation in the graph. The focus in the graph(s) will be either automatically moved (e.g. if the user selects a class within the domain model graph) or manually set within the coupling definition graph (in accordance to the given interaction constraints in *VizCo*). Synchronously, the focus is moved in the domain model graph. The focus only can be set on a class ($moveFocusToClass$) or a property ($moveFocusToProperty$).

$class$, $property$, and $relation$ contain $elementName$, $elementAlias$ and $elementURI$ which represent the terminal rules. $elementAlias$ is an identifier (a variable) for the **instance** domain model element, while $elementName$ is the label of the domain model element (e.g. label of a class) serves the purpose to be displayed within the coupling definition graph. $elementURI$ is a unique reference to the domain model element and is used for the RDF schema to SQL statement conversion. Furthermore, $property$ also contains a $domainClassAlias$ referring to the class, which is the domain of the respective property.

3.6 RDF Query generation

In step 4 of the coupling definition process the action history expression is converted into a domain view description that is based on the RDF query language *RDF Query* [11].

3.6.1 RDF query

This subsection gives a short summary of the relevant parts of RDF Query. RDF Query is a simple query language for RDF developed by IBM. Its main advantages are its clear descriptive character and its representation in XML. The RDF Query model mainly contains the following elements:

1. The query itself is represented by an $rdfquery$ element and contains $from$, $union$, $intersection$, and $difference$ (to be Relational Algebra compliant).

2. Element $from$ contains $select$ and $order$. The $select$ element defines the RDF Schema elements (i.e. class and property) to be included within the Coupling definition. At this time an ordering is not necessary within the Coupling definition, thus $order$ will not be used.

3. Within $select$ we have the $selectProperty$[1], $condition$, and $group$ element. $group$ is not relevant for our work.

4. $condition$ contains $selectProperty$-$selectProperty$ relations or $selectProperty$-$String$ relations. These can be combined with logical operators not, and,and or.

5. A $property$ has the attributes $name$ (which is in our case the URI of the $selectProperty$) and $alias$. To keep the $alias$ of a $property$ is not adequate, since we have to know the domain class $alias$ of the $property$ in order to convert the RDF query into a SQL statement.

3.6.2 Generation of the RDF-based View Description

Before starting the generation process the action history is reduced by a preprocessor. For the RDF view generation not all actions in the history are relevant. add actions and the $emptyGraph$ action are, of course, relevant for the generating function. Since $moveFocus$ does not

[1]We chose the name $selectProperty$ instead of $property$ (as it is defined within the RDF Query specification) to keep them apart.

	M'_{class}	M'_{prop}	$M'_{relprop}$	M_{sel}	M_{cond}
$op1$	$\cup M'_{class}$	-	-	$\cup M'_{class}$	-
$op2$	-	$\cup M'_{prop}$	-	$\cup M'_{prop}$	-
$op3$	-	-	$\cup M'_{relprop}$	-	-
$op4$	-	-	-	-	$\cup M_{cond}$
$op5$	-	$\cup M'_{prop}$	-	-	-
$op6$	-	$-M'_{prop}$	-	-	-

Table 2. Operations and sets of $R(C)$

manipulate the resulting coupling definition at all and $withdrawFromGraph$ can be considered as a "undo" action, both types of actions are eliminated from the action history. The $moveFocus$ actions may be simply deleted. To remove the withdraw actions and the corresponding add actions, the preprocessor makes the following steps. First step is to find the next $withdraw$ expression. Thereafter, the corresponding add is identified and both expressions are removed. The preprocessor repeats these steps, until all $withdraw$ expression are removed. So the relevant rules $addClassToGraph$, $addPropertyToGraph$, $addRelationToGraph$, $addCouplingFlagToGraph$ (either with option=yes or option=no), $addConditionToGraph$ are remaining.

In order to make the resulting domain view definition format selfcontained, we slightly modify the RDF query format. We add the part of the domain model underlying the view definition as an supplementary part to the RDF query expression used for view definition. Both parts together form a complete view definition and are generated from the action history expression. For generating these two parts a coupling generation function c is defined, which produces a RDF view definition document based on the expression of the action history language.

Let S_c= $(M'_{class}, M'_{prop}, M'_{relprop})$ be the submodel of the domain model S_{RDF} to be included into the domain view definition. S_c contains the domain model elements which are referred by the action history expression after eliminating the withdraw actions.

The second part of the RDF view definition is the RDF query defining the conditions for the view. For this purpose we only use the selection and the condition element of RDF Query (plus the from element for referring to the domain model). The query Q_{RDF} may thus be represented by a tuple consisting of a set M_{sel} representing the selection elements and a set M_{cond} representing the condition elements of the query ($Q_{RDF} = (M_{sel}, M_{cond})$). The range of c, thus, is defined as follows. *Definition D10*:

$$R(c) = \left\langle \begin{matrix} S_c \\ Q_{RDF} \end{matrix} \right\rangle = \left\langle \begin{matrix} M'_{class} \\ M_{prop} \\ M'_{relprop} \\ M_{sel} \\ M_{cond} \end{matrix} \right\rangle, \text{ with } M'_{class} \subseteq M_{class}, M'_{prop} \subseteq M_{prop}, \text{and } M'_{relprop} \subseteq M_{relprop}.$$

The domain of the function c is the RDF domain model S_{RDF}.

So, the generating function c has to analyze the reduced action history and has to create the five sets $M'_{class}, M'_{prop}, M'_{relprop}, M_{sel}$, and M_{cond}. Table 2 summarizes the rules for creating these sets starting from five empty sets. The first column contains the possible elements of the action history and the first row contains all sets of $R(c)$. The table entries specify the impact for the respective set, when the respective action is found in the action history. For the pair $addClassToGraph$ and M'_{class}, for example, the action in the action history is $addClassToGraph(graph, c)$ and the resulting set operation is $M'_{class} := M'_{class} \cup c$, which is denoted as $\cup M'_{class}$ in the table. Parameters are omitted since they are obvious. "-" means "no action" (the set stays as it is).

3.7 Query conversion

The last step within the Coupling Definition Process is the conversion of the RDF view definition into SQL query expressions.

There are three parts within the SQL structure, which are relevant for the conversion process. These are $sql : select$[2] (where requested $columns$ are specified), $sql : from$ (where requested $tables$ are specified), and $sql : where$ (where requested $sql : conditions$ are specified). Instead of $sql : select$ there is also $sql : insert$, $sql : update$, or $sql : delete$ for the propagation of changes.

Starting point of the conversion is the RDF view definition produced by the previous step of the coupling definition process. This view definition contains RDF Query select elements (we refer to them as $rdfq : select$) and RDF Query condition elements ($rdfq : condition$). $rdfq : select$ contains either a $property$ or a $class$. Based on definition D9 we define the inverse function $g_{smap}(S_{RDF})$:

Definition D11:

$$\vec{g}_{smap(S_{RDF})} = \vec{f}^{-1}_{smap}\left(\left\langle \begin{matrix} M_{class} \\ M_{prop} \\ M_{relprop} \end{matrix} \right\rangle\right) = \left\langle \begin{matrix} f_{class}(M_{class}) \\ f_{prop}(M_{prop}) \\ f_{relprop}(M_{relprop}) \end{matrix} \right\rangle = \left\langle \begin{matrix} M_{tab} \\ M_{col} \\ M_{rel} \end{matrix} \right\rangle$$

Converting RDF query expressions into SQL expressions all $property$ elements contained in a $rdfq : select$ are inserted into $sql : select$ as a $column$ resulting from the conversion with f_{prop}. The $alias$ of the domain $class$ related to the property is inserted as $table$ into the $sql : from$ clause. So, if we concatenate the $column$ and $table$ elements with the corresponding $alias$ we can build up the SQL statement accordingly. As an example, we consider two $property$ elements $prop1$ and $prop2$. Both have $alias :=' c2'$. Furthermore, we assume a $class$ element $classX$ with the $alias :=' c2'$. With D10 we get $col1 := f_{prop}(prop1)$,

[2] We assume an XML representation of SQL statements that is marked by the namespace prefix sql.

$col2 := f_{prop}(prop2)$, and $tabX := f_{class}(classX)$. The following SQL statement will be generated: $select\ c2.col1, c2.col2\ from\ tabX\ c2$. If another $class$ element $classY$ ($tabY := f_{class}(classY)$) which is related to $classX$ has been added to the query, also a $relproperty$ element $rp1$ is included [3] and will be transformed into a $relationship$ element $rel1 := f_{relprop}(rp1)$.

Using the inverse function of f_{rel} defined in D6 $rel1$ becomes $rel1 = (< foreignkey >, < foreignkeytab >, < primarykeytab >) = (fk1, tabX, tabY)$ and the SQL statement will be extended with $tabY\ c3$ (considering $tabY\ alias = c3$). The inverse function of function f_{tab} defined in D4 returns the $primarykey\ pk3$ of $tabY$. We are now able to specify the natural Join of $tabX$ and $tabY$ and we get $select\ c2.col1, c2.col2\ from\ tabX\ c2, tabY\ c3\ where\ tabY.pk3 = tabX.fk1$.

Now only $rdfq : condition$ remains to be converted. We can easily translate the $rdfq : condition$ into a $sql : condition$ by using the inverse functions of the functions defined in D4-D6. After translation, the $sql : condition$ can be added to the SQL statement. As boolean operator to connect we choose and.

As we illustrated in chapter 3.1, the user interface, as an example application of the coupling, gets all information to retrieve and modify the coupled data records. So far we only covered data selection.If modified data in the user interface have to be synchronized with application data, an additional mapping is required. For synchronization the RDF query as well as the modified result set RS_m will be available.

For space reasons we only give a short description on how this case is handled by the coupling processor. The processor takes the original result set RS_o, which is either cached or retrieved again with help of the SQL statement, and compares it with RS_m. If there are records within RS_m and not within RS_o, these records have to be inserted. Supported by the definitions D1-D11 it is possible to split the generated SQL statement for having a separate insert statement for each $table$. In the inverse case there are records within RS_o and not in RS_m. and a $delete$ statement has to be generated. If the record of RS_m has the same identifier as the record of RS_o, but at least one value is different, then this implies an update statement.

4 The *VizCo* Tool

4.1 Web Application Development Framework

Our Web application development framework aims at supporting the development and evolution of Web applications (and their user interfaces) in the area of information, content and knowledge management in a flexible and user-friendly way. In doing this we follow a meta-design approach [6] by integrating an authoring tool suite as integral part into the system itself. The tools of this suite are used to set up and customize applications as well as to adapt them to evolving business process requirements. It is the goal of the meta-design approach to actively involve part of the users into the design and development process in order to make their (domain) expertise usable in the system in a more direct way [14]. The underlying development paradigm is model-based: domain model components, user model, user interface model, etc. are set into relationship to each other in the design process.

The Form Dialog Manager, mentioned in section 3.1, is just one of the authoring tools in the tool suite. Other authoring tools enable e.g. setting up menus, building up taxonomies [15], classifying content as well as design-related information objects, and controlling the flow of the business process in the application.

The Web application development framework is based on a flexible and extensible, component-based architecture. The implementation is database independent and uses the Web service paradigm for component interaction in order to improve flexibility, extensibility, and reusability. Interaction with other system components is facilitated by an extensive use of standards like XML, RDF, SOAP, XForms, etc.

The prototypical implementation of the Web application framework has been successfully used to set up an operational e-Learning applications [8] as well as a system for supporting trade fair business processes [14]. Currently, a new version of the Web application development framework is under development with an extended and improved set of system authoring tools.

4.2 *VizCo* Use Cases

The user interface of the *VizCo* tool consists of a window with two parts. The left part of the window contains the SRG of the domain model and the right side shows the current state of the composed coupling definition graph (see figure 5). Starting point of a *VizCo* session is an empty coupling definition graph (or the result graph of the previous coupling definition session).

The notion of the "current focus" is an important concept in the interaction with *VizCo*. This concept is e.g. used to ensure that the resulting coupling definition graph is connected: In each step only those elements in the domain model can be selected that are connected with the focus element in the graph.

After selecting an element on the left side (in the domain model), *VizCo* moves the focus on both sides to the selected element. A selection is only possible for class and property

[3] If there is no $relproperty$ for a pair of $class$ elements the result would be a Cartesian product.

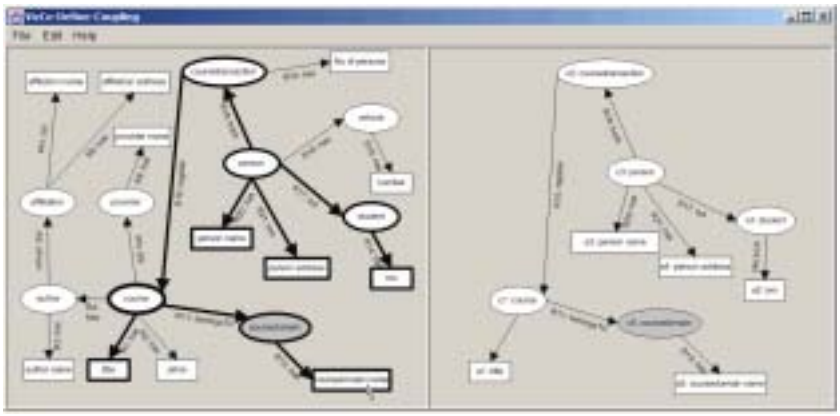

Figure 5. Snapshot of *VizCo* while defining a Coupling

elements. On selection of an element (class or property) the arc connecting the selected element with the current focus element is automatically inserted into the coupling definition graph and is selected for inclusion into the domain view. Only in cases of disambiguities (two arcs connecting the same two elements) user interaction is required. Rel-properties, which are represented by an arc in the SRG, are, thus, not directly selected by the user.

When a class is selected in the domain model graph (by mouse click), a named instance of that class appears within the right part of the window (thus this instance becomes part of the domain view). To be more precise, this is rather a placeholder for possible instances in the view than an actual instance. Properties are selected in the same way as classes for inclusion into the domain view. As discussed in the context of the action history language, two cases have to be distinguished here:

1. The user wants to define one or more conditions for the selected property. In this case the respective property is selected in the coupling definition graph and a condition definition dialog is opened via the right mouse button.

2. The user wants to decide, if the selected property becomes part of the domain view. In this case, the user opens the coupling flag dialog and decides about the inclusion.

In order to withdraw a class or property from the coupling definition graph, the respective element is selected on the right side of the window and the option withdraw is chosen from the menu showing up on a right mouse click. The condition for withdrawing an element have already been discussed in section 3.5.2.

If two properties are used to connect their domain classes (no direct relationship between the classes) via the values of the properties a special arc is inserted into the graph to keep the graph connected and to visualize the link defined between the two properties. To define such a connection, the second property is selected, when the focus is on the first property. This inserts the property and the class, which is the domain of the property into the coupling definition graph and automatically opens a condition definition dialog. When the condition definition dialog is completed, the respective link is inserted into the graph.

5 Conclusions and Future Work

In this paper we presented *VizCo*, a tool for visually defining couplings of Web application elements like user interface components with a domain model and indirectly with the underlying application data. The used domain model is automatically extracted from the application data schema and possibly further enriched by the user. We described the steps of the coupling definition process at design and runtime. Furthermore, we discussed the mappings and representation formats that are necessary to cover the various steps of the coupling definition process.

A first prototype of the Web application framework and its application system authoring tool suite including *VizCo* is implemented. However, we are still working on improvements of *VizCo* and the other tools. In more details, future work in the further development of *VizCo* focuses on the following issues.

The mappings approach defined for *VizCo* will be extended in order to handle RDF in a more comprehensive way. A mapping and integration of ”native” support for RDF content management and query language processing will be investigated. This will facilitate the management of user defined enrichments of the domain model (e.g. by subclass relationships). A further challenge of this integration is the adequate mapping of the semantic of such enrichments onto the SQL queries. This issue will be addressed in the next version of the *VizCo* tool.

Currently we are using IBM’s RDF query as the basis for an intermediate exchange format for coupling definitions. As soon as an RDF query language standard becomes available, the tool and its mappings will be adapted to this standard.

A further area of future work is the exploitation of *VizCo*

in other places of the Web application development framework. We already integrated *VizCo* with the Form Dialog Manager to enable a coupling between the domain model and elements of a form based user interface. As next goals we plan to integrate *VizCo* with an data analysis engine enabling the user friendly customization of the engine and with an information visualization framework enabling the flexible coupling of visualization elements with the data of the application domain (via the domain model). Besides an extension of the Web application development framework we also expect further input for the improvement of the *VizCo* tool itself from these integration activities.

References

[1] M. Angelaccio, T. Catarci, and G. Santucci. Qbd*: A graphical query language with recursion. In *IEEE Transaction on Software Engineering*, volume 16 of *10*, 1990.

[2] T. Berners-Lee, J. Handler, and O. Lassila. The semantic web. *Scientific American*, Special Issue on "Intelligent Systems/Tools In Training And Life-Long Learning", 2001.

[3] Chen. The entity-relationship model: Towards a unified view of data. In *ACM Transactions on Database Systems*, 1976.

[4] M. P. Consens, I. F. Cruz, and A. O. Mendelzon. Visualizing queries and querying visualizations. *SIGMOD Record*, 21(1):39–46, 1992.

[5] M. Dubinko, L. Klotz, R. Merrick, and T. V. Raman. Xforms 1.0 specification-w3c candidate for recommendation, 2002.

[6] G. Fischer and E. Scharff. Meta-design: Design for designers. In *Proceedings in DIS2000 Conference*, 2000.

[7] J. Foley, C. Gibbs, W. Kim, and S. Kovacevic. A knowledge-based user interface management system. In *Readings in Intelligent User Interfaces*, San Francisco, 1998. ACM Press.

[8] M. Fuchs, C. Muscogiuri, C. Niederée, and M. Hemmje. An open framework for integrated qualification management portals. In *Thirteenth International Workshop on Database and Expert Systems Applications*, 2002.

[9] K. Goldman, S.A.Goldman, P. Kanellakis, and S. Zdonik. Isis: Interface for a semantic information system. In *ACM-SIGMOD Intl. Conf. on Management of Data*. ACM Press, 1985.

[10] G. Karvounarakis. Rdf query languages: A state-of-the-art, 1999.

[11] A. Malhotra and N. Sundaresan. Rdf query specification. QL'98 - Query Languages 1998, 1998.

[12] M. Marchiori and J. Saarela. Query + metadata + logic = metalog. QL'98 - Query Languages, 1998.

[13] L. Miller. Rdf query by example. QL'98 - Query Languages, 2002.

[14] C. Muscogiuri, C. Niederée, M. Hemmje, and M. Fuchs. Towards meta-design for e-business: Experiences & challenges. In *Proceedings of the Human Computer Interaction Consortium Winter Workshops (HCIC 2002)*, 2002.

[15] C. Niederée, C. Muscogiuri, and M. Hemmje. Taxonomies in operation, design, and meta-design. In *Proceedings of the International Workshop on Data Semantics in Web Information Systems (DASWIS)*. IEEE CS, 2002.

[16] A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. In *Proceedings of the Intelligent User Interfaces*. ACM Press, 1999.

[17] A. R. Puerta. A model-based interface development environment. In *IEEE Software*, volume 14 of *4*, 1997.

[18] A. R. Puerta, Eriksson, Gennari, and Musen. Model-based automated generation of user interfaces. In *Readings in Intelligent User Interfaces*, San Francisco, 1998. ACM Press.

[19] M. Sintek and S. Decker. Triple—a query, inference, and transformation language for the semantic web. International Semantic Web Conference (ISWC), Sardinia, 2002.

[20] C. Wiecha, W. Bennett, S. Boies, J. Gould, and S. Greene. A tool for rapidly developing interactive applications. In *Readings in Intelligent User Interfaces*, San Francisco, 1998. ACM Press.

[21] M. Zloof. Query-by-example: Operations on the transitive closure. Research RC5526, IBM, Yorktown Heights, 1976.