

Relational Data Tailoring Through View Composition^{*}

Cristiana Bolchini, Elisa Quintarelli, and Rosalba Rossato

Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza Leonardo da Vinci, 32 – 20133 Milano, Italy
{bolchini,quintare,rossato}@elet.polimi.it

Abstract. This paper presents a methodology to derive views over a relational database by applying a sequence of appropriately defined operations to the global schema. Such *tailoring* and *composition* process aims at offering personalized views over the database schema, so as to improve its ability to support the new needs of customers, support evolutionary software development, and fix existing legacy database design problems. The process is driven by the designer's knowledge of the possible operational contexts, in terms of the various dimensions that contribute to determine which portions of the global schema are relevant with respect to the different actors and situations. We formally introduce some operators, defined on sets of relations, which tailor the schema and combine the intermediate views to derive different final views, suitable for the different envisioned situations. The application to a case study is also presented, to better clarify the proposed approach.

1 Introduction

The development of complex systems dealing with huge amounts of information requires a careful design phase, where all actors need be identified together with all the elements that may determine what portion of the entire data they should have access to in the various situations. Actually, this is not so much a matter of privacy and security, as of efficiency and usability, when too much information may cause confusion rather than knowledge.

In the relational scenario, the above problem consists in designing appropriate views to provide different, selected data access to portions of the entire data schema. Yet, if the number of elements, or *dimensions*, determining the subset of data interesting for each given *context*, is high, the designer's task may be quite complex, since all possible contexts for a target scenario have to be examined and, for each one of them, the associated view has to be defined.

The aim of our proposal is the definition of a systematic methodology that first supports the designer in identifying, for a given application scenario, the dimensions for characterizing the context and their possible values, and then, once the designer has identified *partial views* with respect to such context dimensions, combines them to produce the final views, one for each possible context. We refer to this view production process as *schema tailoring*, since it amounts to *cutting out* the appropriate data portion which fits each possible context.

^{*} This research is partially supported by the MIUR projects ESTEEM and ARTDECO.

Our claim is that, even for small application scenarios (such as the running example introduced below), where the number of dimensions driving the tailoring of the schema is quite small, the resulting number of possible contexts is rather high: as a consequence, the adoption of the proposed approach not only helps the designer consider all possible situations, with no omission, but also automatically performs the (computationally significant) view generation task.

The rest of this paper is organized as follows. Section 2 presents the overall scenario, together with the motivations of our work and its contribution with respect to related studies. Section 3 introduces the view definition methodology and the necessary operators supporting view composition. The application to a real example is discussed in order to validate the approach and its feasibility. Future work and concluding remarks are reported in the last section.

2 Background, Motivations and Rationale

The proposed methodology aims at providing the database designer with a systematic approach to view design, in a scenario where different database users, situations and other elements (altogether called dimensions) affect the choice of the subset of data to be considered interesting and made available to such actors.

In order to define the methodology, three elements are necessary:

- a) a model expressing the dimensions, and their values, driving schema tailoring. The model should be able to capture the dimensions as well as all the possible contexts deriving from their instantiations;
- b) a strategy for identifying for each dimension, independently of the others, a relevant portion of data on the entire schema – the so-called *partial views*; and
- c) a set of operators for combining the partial views to derive the final view(s) associated with each context.

In previous work [2,4], we have presented a context model, called *Context Dimension Tree*, which takes care of point (a), whereas in this paper we focus on the last two points, where the Context Dimension Tree is used to create and compose views for data tailoring purposes.

Let us consider the case of a real estate agency storing data related to its agents, estates and owners and wanting to support the agents in their work (in office and also on-site, when they take prospective clients to visit the properties), and to promote its business on the Web. The agency database stores also information on the closed contracts (rents and sales), to provide the supervisor with an overview of the situation. The scenario, although simple, has four different actors: supervisors, agents, buyers and sellers. Supervisors must have a global view of the business, in terms of the estates the agency deals with, and of the agents' work. Agents are in charge of the visits and of acquiring new estates for the agency portfolio. The corporate database is also the source for a Web site to promote business, and views need be designed to provide all and only the necessary data. In this paper we base our considerations on the relational data model, thus the real estate database is composed by the tables reported in Fig. 1.

OWNER(IdOwner, Name, Surname, Type, Address,City, PhoneNumber)
 ESTATE(IdEstate, IdOwner, Category, Area, City, Province, RoomsNumber,
 Bedrooms, Garage, SquareMeters, Sheet, CadastralMap)
 CUSTOMER(IdCustomer, Name, Surname, Type, Budget, Address, City, PhoneNum)
 AGENT(IdAgent, Name, Surname, Office, Address,City,Phone)
 AGENDA(IdAgent, Data, Hour, IdEstate, ClientName)
 VISIT(IdEstate, IdAgent, IdCustomer, Date, ViewDuration)
 SALE(IdEstate, IdAgent, IdCustomer, Date, AgreePrice, Status)
 RENT(IdEstate, IdAgent, IdCustomer, Date, RatePrice, Status, Duration)
 PICTURE(IdPicture, IdEstate, Date, Description, FileName)

Fig. 1. The database schema

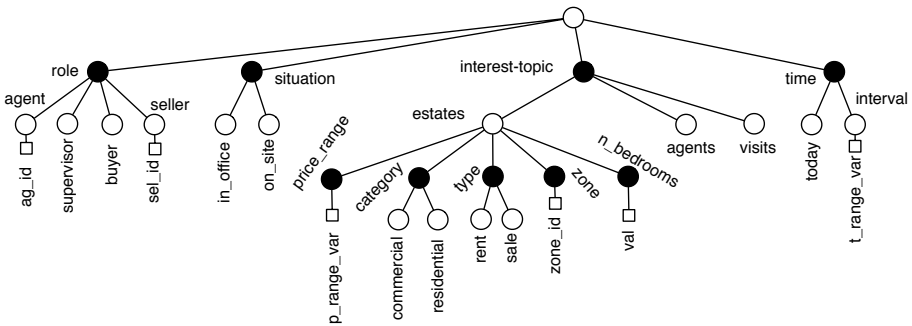


Fig. 2. The Context Dimension Tree

The first step of our approach consists in specifying the context dimensions characterizing the possible contexts in this application scenario, also listing the expected values, according to the methodology presented in [4]. More precisely, our context model, the Context Dimension Tree, is an extension of the context model presented in [6], and is used to systematically describe the user needs, and to capture the context the user is acting in. It plays a fundamental role in tailoring the target application data according to the user information needs. Once the possible contexts have been designed, each must be connected with the corresponding view definition. In this way, when the context becomes current, the view is computed and delivered to the user. The various ways to operate such design-time connection are the subject of this paper.

Fig. 2 shows the Context Dimension Tree we have elaborated for our real estate agency, better explained below.

2.1 Background

The Context Dimension Tree is a tree $\mathcal{T} = \langle N, E, r \rangle$, whose nodes are the context dimensions and their values. The set of nodes N is partitioned into the subsets N_D , N_C and N_A . The two main types of nodes are: *dimension nodes*, N_D , black, and *concept nodes*, N_C , white. N_A is the set of *attributes*, whose explanation we postpone for the moment.

The root's children are the *top dimensions*, which capture the different characteristics of the users and of the context they are acting in; in our example, the top dimensions are the database user's *role*, his or her *situation*, the current *interest topic* and the *time span* of interest. A dimension value can be further analyzed with respect to different viewpoints (sub-dimensions), generating a subtree in its turn. For example, the interest topic "estates" can be analyzed w.r.t. the price range, the category, the business type, the zone and the number of bedrooms.

The root r of the tree is a concept node: it models all the possible contexts, thus represents the entire dataset, before tailoring.

The edge set E is partitioned into the two subsets E_R and E_A : E_R contains *sub-element of arcs*, whereas E_A contains *attribute of arcs*.

Coherently with the meaning of dimension and concept nodes, each "generation" contains nodes of the same color, and colors are alternated while descending the tree: $\forall e = \langle n, m \rangle \in E_R$, either $n \in N_D \wedge m \in N_C$ or $n \in N_C \wedge m \in N_D$; i.e., a dimension node has concept nodes as children and a concept node has dimension nodes as children.

We consider now *attribute nodes*: $\forall n \in N_A, \forall e = \langle m, n \rangle \in E_A, m \in N_D \cup N_C$; i.e., the parent of an attribute node is either a dimension node or a concept node. In the first case, the attribute represents a *parameter*, while in the second case it represents a *variable*. Consider the sub-dimensions of estates: while *category* and *type* have white children (i.e., their values), *price_range* only features a small square, an em attribute node; this attribute is a selection parameter whose instances represent the possible values of the *price_range* dimension, e.g., a price interval. Thus, dimension branching can be represented in both ways: either by explicitly drawing the values as white nodes, or by indicating a parameter whose instances are the dimension values. This does not modify the expressive power of the model: rather, it makes it more readable, and more usable for the designer.

Also white nodes may feature a variable indicating how to select a specific set of data instances. As an example, consider the *ag_id* variable associated with the *agent* role: it is an identifier used, at run-time, to select the data related to a specific agent.

Attribute nodes are *leaves*, i.e., $\forall n \in N_A, \neg \exists m \text{ s.t. } \langle n, m \rangle \in E$. Moreover, according to the meaning of attribute nodes, each attribute node is an *only child*: $\forall n \in N_A, \forall e = \langle m, n \rangle \in E_A$, if $\exists e_1 = \langle m, n_1 \rangle \in E$ then $n = n_1$.

Dimension nodes without concept children *must have an attribute child*, i.e., $\forall n \in N_D$ such that $\neg \exists e = \langle n, m \rangle \in E_R$ then $\exists e_1 = \langle n, m_1 \rangle \in E_A$. Indeed, black nodes must necessarily feature either white children or the parameter node, whose values are those of the corresponding subdimension. Thus, leaf nodes can only be either attribute nodes or white nodes.

Finally, in order to simplify the discussion, we also introduce, although redundant, two more sets: $\overline{N_D}$, which identifies dimension nodes that have an attribute child (i.e., $\overline{N_D} = \{n \in N_D \mid \exists e = \langle n, m \rangle \in E_R\}$); *context elements* c are in $N_C \cup \overline{N_D}$, which identifies either a concept node (white node), such as the *supervisor*, or a dimension node (black node) with a parameter (such as *price_range(\$p_range_var)*).

The tree is formalized in [2], and can be specified in several ways; in particular we propose an ontological representation in the OWL language [12], and an XML representation based on a DTD [5].

The complex structure originating from the composition of the context elements of a Context Dimension Tree is called a *context* or *configuration*, and determines a portion of the entire data set, i.e., a *chunk*, specified at design-time, to be selected later, at run-time, when the corresponding context becomes active. A *context* is expressed as a tuple of context elements, each of them described with the form

```
dim_name : value
```

where *dim_name* is the name of a (sub-)dimension, and *value* is a value for that dimension. The values for each context element may be at any level in the tree, thus allowing for different levels of granularity. Note that sibling white nodes are *mutually exclusive* since they represent orthogonal concepts, while sibling black nodes represent the different (sub-)dimensions which define a concept. Therefore, when building a configuration, for each top dimension, at each level, only one white node among each set of siblings, and any number of black siblings may be included.

Note that a context can lack some dimension value(s): this means that those dimensions are not taken into account to tailor data, i.e., the view corresponding to that configuration does not filter the data for these dimension(s).

Let us consider the situation of an agent ($\$ag_id="XYZ"$) ready to take prospective buyer clients to visit the residential estates properties located in the "Piola" area ($\$zone_id="Piola"$). The current context (or configuration) *C* is composed by the nodes

```
{ role : agent("XYZ"),
  category : residential,
  type : sale,
  zone : zone("Piola"),
  situation : on_site,
  time : today(getdate())}
```

wherefrom we have to derive the specification of the information related to the interesting properties.

Such configuration must be associated at design time with the (definition of the) piece of data which must become available when *C* occurs, and the same must be done for each possible (and reasonable) configuration. Note that the instantiation of the zone is based on the value "Piola", appropriately fed to the system at run time: this is an example of use of a *variable*, to be suitably used by the run-time system.

The designer can also express constraints or preferences on the possible combinations of the context elements; in fact, not all of them make sense for a given scenario, thus, if we combinatorially generate the complete set, many configurations must be discarded. Hence, the tree model is enriched by introducing *forbid constraints*: they allow the context designer to specify configurations that are not significant, thus discarding those that would represent semantically meaningless context situations or would be irrelevant for the application. For example, such a constraint can be established to forbid that the buyer role's context contains interest topics related to the agents' administrative data.

The introduction of these constraints, not further discussed here, allows the designer to represent with the Context Dimension Tree all and only the significant contexts for the given application scenario.

2.2 Goal and Contributions

Once the Context Dimension Tree has been designed, the subsequent work of the designer can take two different directions, one more time-consuming but allowing for a more precise view definition, the second one more automatic but more prone to possible errors, thus to be verified a-posteriori.

When the first strategy is adopted, once the configurations are combinatorially derived the designer associates each of them with the corresponding portion of the information domain schema. This is called *configuration-based mapping*, and can be done by directly writing a query in the language supported by the underlying database, or by selecting these portions by means of a graphical interface which will derive the corresponding view. This process is already supported by a tool we have developed [3]; however, as it can be expected, even after excluding the meaningless configurations by means of the appropriate constraints, a medium-size Context Dimension Tree originates a huge number of configurations (e.g., in our running example there are 816 significant contexts), thus the task of associating each significant context with its view is unpractical. Moreover, if the context model changes, e.g., if a new concept is inserted, a high number of new configurations will have to be generated from its combination with the other concepts, and the designer will have to redefine all the mappings for these. Similarly, also a change in the underlying database schema will require a complete revision of all derived views.

To overcome the limitations of this first approach, in this work we formalize a compositional strategy, called *node-based mapping*, whereby the designer selects the schema portion to be associated *with each context element* and then the system will combine them within the configurations. In this paper we first define the possible policies to associate schema portions with context elements, then introduce the appropriate logical operators to derive the view associated with a configuration. Different combination policies will also be presented, involving the use of proper operators, such as intersection or union of schemata and instances, to produce the final result.

The proposed approach to relational data tailoring taking into account all these aspects has been implemented in another tool we developed, supporting the user in all steps of the methodology.

2.3 Related Work

In the past years, the view definition process has been exhaustively investigated in the database literature; the first models and view-design methodologies were for relational databases [8].

When focusing on the relational model of data, a few solutions have been presented to introduce the notion of context in the data being modeled, by means of additional attributes explicitly defining the context (e.g., space and time characteristics) the data belongs to.

In [15] the authors present a Context Relational Model as an extension of the relational one: the notion of context is treated as a first-class citizen that must be modeled

and can be queried by using standard SQL expressions. A lot of attention is devoted to the investigation of instance changes on the basis of context dimension values. This is a significant difference with respect to our approach, where the context model is aimed at data tailoring, that is the extraction of the relevant portion of a relational schema for a given context.

In [16] the authors introduce a context-aware system that uses context, described as a finite set of context parameters, to provide relevant information and/or services to its users. The proposed system supports preference queries whose results depend on context. Users express their basic preferences by associating a degree of interest between attribute values and instances of a context parameter; basic preferences are combined to compute aggregate preferences, that reflect the users' selection with respect to a more complex notion of context. The approach is more focused on personalization of query answering, however, the context-dependent view definition process is not performed at design time, but is guided by the available user's preferences.

The success of XML has motivated, in the recent years, the necessity of proposals that are independent of the underlying data model: the problem of view definition for XML has been formalized in [1,7], and some other works in the XML area are [10,20]. Some other view models based on ontologies have been proposed in [17,18] in the Semantic Web Area.

All the above cited models follow the classical notion of view extraction process; thus, the state-of-the-art in view design needs to be extended to consider also user's activity, tasks, and intentions: the user's context.

In [9,13,14,19] the authors propose formal frameworks for specifically addressing the problem of view design for XML and ontological data sources, guided by a notion of context; in [14] some conceptual operators are formalized for constructing conceptual context dependent views on XML data; the notion of context is related to facts of data warehouses and the operators extract information related to a single specific fact, without combining different views. In our opinion, the notion of context we use in this paper is much more articulated (mutual exclusive and orthogonal dimensions are modeled) and thus, we need to define operators both to extract and to combine partial views; the composition step is required to obtain views suitable to represent the relevant portion of data for a notion of context that depends on more than one perspective.

3 View Definition

In this section we describe the way to derive a view for each possible context, by starting from the representation of the application context by means of a Context Dimension Tree and by adopting a *node-based mapping* strategy. The process is composed by two main steps, detailed in the rest of the section:

- *Relevant area assignment*: a partial view, expressed as a set of relational algebra expressions, is associated with each context element;
- *View composition by algebraic operators*: the previously defined partial views, associated with the context elements in a configuration, are properly combined to automatically obtain a final view defining all the information relevant for that configuration.

3.1 Relevant Area Assignment

In this section we formalize the way to associate a partial view with each context element c ($c \in N_C \cup \overline{N_D}$). In this step, the designer annotates the Context Dimension Tree by labeling each context element with the partial view assigned to it via the mapping

$$\mathcal{R}el : N_C \cup \overline{N_D} \rightarrow \wp(\mathcal{V}) \quad (1)$$

where \mathcal{V} is the set of views on the global database, and each partial view $V \in \wp(\mathcal{V})$ is a *set of relational algebra expressions* e_i , each one of them recursively defined as follows:

$$e_i \stackrel{\text{def}}{=} R_j | \pi_{att}(e_i) | \sigma_{\theta}(e_i) | e_i \bowtie_{\theta} e_j | e_i \bowtie_{\theta} e_j \quad (2)$$

where R_j is a relation belonging to the global schema, and att and θ are shorthands to denote a set of attributes and a boolean condition, respectively.

Relevant Area Policies

When tailoring the relevant area (partial view) for a given context element c , two policies have been identified:

- a) **Maximal relevant area:** the partial view V for c contains all the schema portions that *might be related* to that element. For example, the maximal relevant area for the estate would be

$$\mathcal{R}el(\text{estate}) = \{\text{ESTATE}, \text{OWNER}, \text{VISIT}, \text{SALES}, \text{RENT}, \text{AGENDA}, \text{PICTURE}\}$$

thus, the partial view associated with the *estate* element includes all the information that is related, in some way, to estates; that is, besides the *ESTATE* and *PICTURE* tables, which contain all and only information on the estates, the view also contains all the further information related to properties, including the *OWNER*, *VISIT*, *SALES*, *RENT*, and *AGENDA* tables.

- b) **Minimal relevant area:** the partial view V of a given context element c contains only the portions of the global schema that *are strictly related* to c . For example, for the *estate* interest topic, the minimal relevant area would be

$$\mathcal{R}el(\text{estate}) = \{\text{ESTATE}, \text{PICTURE}\}$$

which only includes tables *ESTATE* and *PICTURE*.

Independently of the adopted policy, an important consideration on the relationships among partial views associated with context elements must be highlighted. More precisely, the hierarchical structure of the Context Dimension Tree has been designed with the aim of increasing the detail level adopted to select data while descending the subtree related to each context dimension. Indeed, while descending a sub-tree rooted in a dimension, we add details in specifying how to use that dimension to tailor data. Thus, in our opinion it is desirable that the partial view for a node n contain the partial view of each descendant m of n . We say that a node n is more abstract than m , and write $n \prec m$, if and only if $m \in \text{Descendant}(n)$.

Notation. Given a generic relation R , we denote with $\text{Sch}(R)$ its schema, i.e., its name and the set of its attributes. Moreover, given two relations R_i and R_j , we say that $\text{Sch}(R_i)$ is a sub-schema of $\text{Sch}(R_j)$, and write $\text{Sch}(R_i) \subseteq \text{Sch}(R_j)$, if and only if the attributes of R_i are a (sub-)set of the attributes of R_j .

The *containment relationship* between partial views is defined as follows:

Definition 1. Let w and k be context elements and $\mathcal{R}el(w)$, $\mathcal{R}el(k)$ the corresponding partial views. $\mathcal{R}el(w) \subseteq \mathcal{R}el(k)$ if and only if:

$$\forall R_i \in \mathcal{R}el(w) \exists R_j \in \mathcal{R}el(k) \text{ s.t. } \text{Sch}(R_i) \subseteq \text{Sch}(R_j) \quad (3)$$

Based on the philosophy of the Context Dimension Tree and its hierarchical structure to support a refinement of the tailoring criteria as the depth of the tree increases, we introduce the following assumption, used throughout this paper.

Assumption 1. For each pair of context elements n and m in a Context Dimension Tree, if $n \prec m$ then $\mathcal{R}el(n) \supseteq \mathcal{R}el(m)$.

To satisfy Assumption 1, given two context elements n and m in a Context Dimension Tree such that $n \prec m$, and supposing the designer has set $\mathcal{R}el(n) = \{R_1, \dots, R_k\}$, then each relation S_i in $\mathcal{R}el(m) = \{S_1, \dots, S_t\}$, with $t \leq k$, is recursively defined as follows:

$$S_i \stackrel{\text{def}}{=} R_j | \pi_{att}(S_i) | \sigma_{\theta} S_i | S_i \times_{\theta} S_j | S_i \times_{\theta} R \quad (4)$$

where $R_j \in \mathcal{R}el(n)$, and R is a relation of the global database.

Example 1. Let us now explain our assumption about containment of partial views related to two context elements, one element a descendant of the other. The partial view for the estate *interest-topic* is a view containing all the information available in the database regarding properties and related information. Formally, using the maximal area policy, we have:

$$\mathcal{R}el(\text{estate}) = \{\text{ESTATE}, \text{OWNER}, \text{VISIT}, \text{SALES}, \text{RENT}, \text{AGENDA}, \text{PICTURE}\}$$

The partial view for the *residential category* is a set of views further restricting $\mathcal{R}el(\text{estate})$ only to information about the residential estates:

$$\begin{aligned} \mathcal{R}el(\text{residential}) = & \{ \sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}, \text{OWNER} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}), \\ & \text{VISIT} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}), \text{SALE} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}), \\ & \text{RENT} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}), \text{AGENDA} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}), \\ & \text{PICTURE} \times (\sigma_{\text{Category}=\text{"Residential"}} \text{ESTATE}) \} \quad \blacklozenge \end{aligned}$$

Context Dimension Tree Traversal During Relevant Area Assignment

In our opinion and experience, the maximal area policy is intended to assign the widest possible view for each context element of the Context Dimension Tree; consequently, it is more natural for the designer, once this policy is selected, to perform the relevant area assignment phase by navigating the Context Dimension Tree *top-down* (from the root to the leaves). This actually means that the view for the root node is the entire

database, and the partial view for a node m is defined by restricting the partial view of its nearest white ancestor. Thus, this phase is performed by navigating the Context Dimension Tree top-down and by specifying for each node m a view $\mathcal{R}el(m)$ on the (previously defined) view $\mathcal{R}el(n)$ of its parent n , according to the recursive definition reported in Equation 4.

On the contrary, the minimal area policy should be used when the designer prefers to specify more detailed views, i.e., the minimal partial view, for each context element of the Context Dimension Tree; thus, when this policy has been chosen, in order to be adherent with the spirit of the Context Dimension Tree, it is recommended to perform the relevant area assignment phase by navigating the Context Dimension Tree *bottom-up*. This means that the partial views are defined by starting from the leaf context elements and the view of a non-leaf node can be obtained by composing the partial views associated with its children.

The composition is obtained by using the double-union operator (introduced later in the paper) and possibly including additional portions of the global database that the designer considers useful for the more general (non-leaf) context element. For example, when using the minimal policy, $\mathcal{R}el(estates)$ can be obtained by combining $\mathcal{R}el(price_range)$, $\mathcal{R}el(zone)$, $\mathcal{R}el(category)$, $\mathcal{R}el(type)$, and $\mathcal{R}el(n_bedrooms)$.

In the *node-based mapping* approach, whose dynamics is shown in Fig. 3, once each context element has an associated partial view (Relevant Area Assignment Phase of Fig. 3), the next step consists in automatically composing for each configuration its final

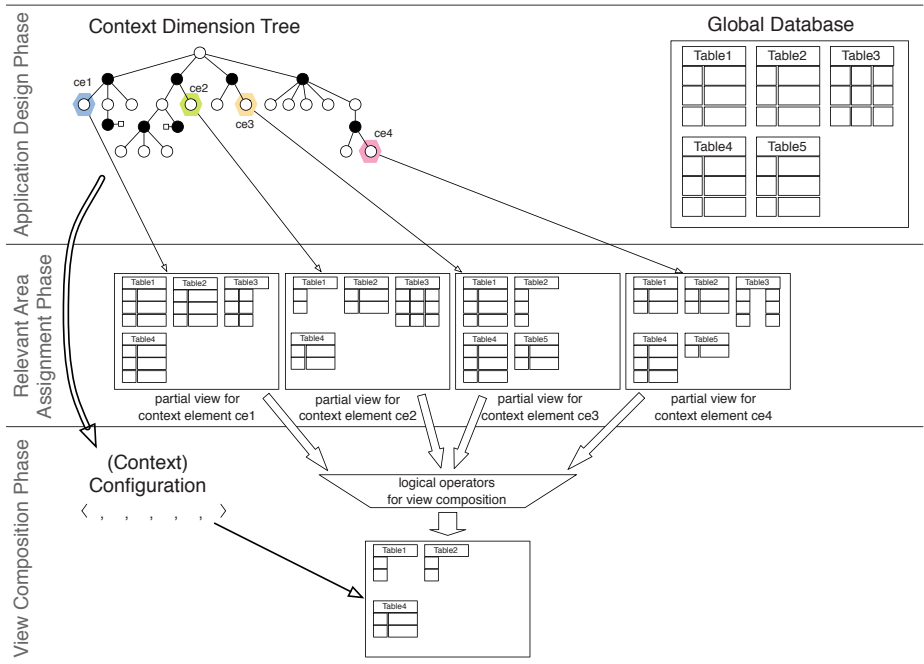


Fig. 3. The node-based mapping strategy

view, computed as a combination of the partial views corresponding to each context element in the configuration (View Composition Phase of Fig. 3). The combination is performed by means of opportunely defined operators acting on sets of views.

This *node-based mapping* approach is more efficient than the *configuration-based mapping* previously developed, yet it is less precise in the definition of the final view associated with a configuration. Thus, we also consider, although not further discussed here, the possibility to integrate additional information in the resulting view, should the designer deem the automatic procedure to have excluded some relevant information.

Example 2. Let us assume the designer has mapped the *buyer role* to a view describing both commercial and residential flats and houses, that is, the *ESTATE* table deprived of the *IdOwner* and *CadastralMap* attributes. Formally:

$$\mathcal{R}el(\text{buyer}) = \{\Pi_Z \text{ESTATE}, \text{PICTURE}\}$$

where

$$Z = \{IdEstate, Category, Area, City, Province, RoomsNumber, Bedrooms, Garage, SquareMeters\}.$$

The *residential category* has a partial view describing residential flats and houses, that is:

$$\mathcal{R}el(\text{residential}) = \{\sigma_{Category="Residential"} \text{ESTATE}, \text{PICTURE} \bowtie (\sigma_{Category="Residential"} \text{ESTATE})\}$$

The view for the configuration $C = \langle \text{role} : \text{buyer}, \text{category} : \text{residential} \rangle$ is obtained by applying a suitable operator for view composition between $\mathcal{R}el(\text{buyer})$ and $\mathcal{R}el(\text{residential})$. \blacklozenge

We have defined two possible operators for view composition, based on the union and the intersection relational algebra operators, discussed in the next section.

3.2 Logical Operators for View Composition

The context-defined views on a relational database may be written by using relational algebra or, equivalently, in SQL. In order to compose different partial views, and automatically gather all and only the information of interest with respect to a specific context, we need to introduce a suite of new algebraic operators which work on *sets of relations*. These operators combine tuples of relations selected by the designer during the partial views definition; therefore, in order to maintain the identity of each tuple during manipulation, we include for each relation in a partial view, its primary key attribute(s). Let \mathcal{A} and \mathcal{B} be two sets of relations, and $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{B}}$ the corresponding sets of relational schemata.

Double Union Operator: The double union $\mathcal{A} \uplus \mathcal{B}$ between two set of relations \mathcal{A} and \mathcal{B} returns the union of (1) the set of all relations R_i whose schema is in $(\mathcal{S}_{\mathcal{A}} - \mathcal{S}_{\mathcal{B}}) \cup (\mathcal{S}_{\mathcal{B}} - \mathcal{S}_{\mathcal{A}})$ and (2) a sort of intersection between R_A and R_B for each pair of relations $R_A \in \mathcal{A}$ and $R_B \in \mathcal{B}$, having a common sub-schema. In this way we preserve selection conditions which have possibly been applied on the same tables in the definition phase of both \mathcal{A} and \mathcal{B} , as we will prove in Theorem 1. Formally:

Definition 2. Let \mathcal{A} and \mathcal{B} be sets of relations.

$$\mathcal{A} \uplus \mathcal{B} = \left\{ R \left| \begin{array}{l} (R \in \mathcal{A} \wedge \neg \exists R' \in \mathcal{B}, \text{Sch}(R) \cap \text{Sch}(R') \neq \emptyset) \vee \\ (R \in \mathcal{B} \wedge \neg \exists R' \in \mathcal{A}, \text{Sch}(R) \cap \text{Sch}(R') \neq \emptyset) \vee \\ (R = \pi_S(R_A) \cap \pi_S(R_B), R_A \in \mathcal{A}, R_B \in \mathcal{B}, S = \text{Sch}(R_A) \cap \text{Sch}(R_B) \neq \emptyset) \end{array} \right. \right\}$$

According to its formal definition, the Double Union Operator is *commutative* and *associative*.

Property 1. Let \mathcal{A} and \mathcal{B} be two sets of relations. For each pair of relations $R_A \in \mathcal{A}$ and $R_B \in \mathcal{B}$ such that $R_A = \Pi_{Z_A}(\sigma_{\theta_A}(R))$ and $R_B = \Pi_{Z_B}(\sigma_{\theta_B}(R))$, with R a relation obtained from the global database, $Z_A \subseteq \text{Sch}(R)$, $Z_B \subseteq \text{Sch}(R)$, the selection conditions θ_A and θ_B are preserved by the application of $\mathcal{A} \uplus \mathcal{B}$.

Proof. R_A and R_B are obtained by considering only those tuples of R that satisfy the selection conditions θ_A and θ_B , restricted on the sub-schemata Z_A and Z_B , respectively. Let $Z = Z_A \cap Z_B$ be the (possibly empty) common sub-schema of R_A and R_B ; then, according to the definition of \uplus , the relation Y , defined as follows, belongs to $\mathcal{A} \uplus \mathcal{B}$:

$$\begin{aligned} Y &= \pi_Z(R_A) \cap \pi_Z(R_B) \\ &= \pi_Z(\pi_{Z_A}(\sigma_{\theta_A}(R))) \cap \pi_Z(\pi_{Z_B}(\sigma_{\theta_B}(R))) \end{aligned}$$

By starting from the fact that $Z \subseteq Z_A$, $Z \subseteq Z_B$ and according to the known equivalence transformation between relational algebra expressions, we can conclude that

$$Y = \pi_Z(\sigma_{\theta_A \wedge \theta_B}(R)) \in \mathcal{A} \uplus \mathcal{B}$$

that is, both conditions θ_A and θ_B are preserved in the resulting relation included in $\mathcal{A} \uplus \mathcal{B}$. \square

Example 3. Assume the partial views of the residential category and the buyer role reported in Example 2 (minimal area policy). The view associated with the configuration $C = \langle \text{role} : \text{buyer}, \text{category} : \text{residential} \rangle$ obtained by applying the Double Union operator between $\mathcal{R}el(\text{buyer})$ and $\mathcal{R}el(\text{residential})$ is the following:

$$\begin{aligned} \mathcal{R}el(C) &= \mathcal{R}el(\text{buyer}) \uplus \mathcal{R}el(\text{residential}) \\ &= \{ \sigma_{\text{Category}=\text{“Residential”}}(\Pi_Z(\text{ESTATE})), (\text{PICTURE}) \times \sigma_{\text{Category}=\text{“Residential”}}((\text{ESTATE})) \} \end{aligned}$$

where $Z = \{ \text{IdEstate}, \text{Category}, \text{Area}, \text{City}, \text{Province}, \text{RoomsNumber}, \text{Bedrooms}, \text{Garage}, \text{SquareMeters} \}$. \blacklozenge

Example 4. Assume the relevant area for the residential category as in Example 3, and for the agent role the minimal relevant area containing his/her data and agenda:

$$\mathcal{R}el(\text{agent}(\$ag_id)) = \{ \sigma_{\text{IdAgent}=\$ag_id}(\text{AGENT}), \sigma_{\text{IdAgent}=\$ag_id}(\text{AGENDA}) \}$$

The view associated with $C = \langle \text{role} : \text{agent}(\$ag_id), \text{category} : \text{residential} \rangle$ obtained by applying the Double Union operator between $\mathcal{R}el(\text{agent}(\$ag_id))$ and $\mathcal{R}el(\text{residential})$ is:

$$\begin{aligned} \mathcal{R}el(C) &= \mathcal{R}el(\text{agent}(\$ag_id)) \uplus \mathcal{R}el(\text{residential}) \\ &= \{ \sigma_{\text{IdAgent}=\$ag_id}(\text{AGENT}), \sigma_{\text{IdAgent}=\$ag_id}(\text{AGENDA}), \\ &\quad \sigma_{\text{Category}=\text{“Residential”}}(\text{ESTATE}), (\text{PICTURE}) \times \sigma_{\text{Category}=\text{“Residential”}}((\text{ESTATE})) \} \quad \blacklozenge \end{aligned}$$

Double Intersection Operator: The double intersection $\mathcal{A} \pitchfork \mathcal{B}$ between two sets of relations \mathcal{A} and \mathcal{B} , applies the intersection operator \cap of relational algebra to the maximal common sub-schemata of the pairs of relations R_A and R_B , belonging to \mathcal{A} and \mathcal{B} , respectively, where either $\text{Sch}(R_A) \subseteq \text{Sch}(R_B)$ or viceversa.

On the result produced by the application of such intersections, the union operator \cup is iteratively applied to pairs of the obtained relations having the same schema, since the Double Intersection can generate more than a single relation with the same schema; this final step aims at removing such duplicates.

Definition 3. Let \mathcal{A} and \mathcal{B} be sets of relations.

a) Let S be computed as follows:

$$S = \{X \mid X = \pi_S(X_A) \cap \pi_S(X_B) \text{ s.t. } X_A \in \mathcal{A}, X_B \in \mathcal{B}, S = \text{Sch}(X_A) \cap \text{Sch}(X_B) \neq \emptyset\}$$

b) $\mathcal{A} \pitchfork \mathcal{B}$ is recursively obtained as follows:

$$\mathcal{A} \pitchfork \mathcal{B} = \left\{ R \mid \begin{array}{l} (R = R_1 \cup R_2, \forall R_1, R_2 \in S \text{ s.t. } \text{Sch}(R_1) = \text{Sch}(R_2)) \vee \\ (R \in S \wedge \nexists R' \in S \text{ s.t. } \text{Sch}(R) = \text{Sch}(R')) \end{array} \right\}$$

According to its formal definition, the Double Intersection Operator is *commutative* and *associative* too.

Example 5. The partial view associated with the agent *role* in case of maximal area policy is

$$\mathcal{R}el(\text{agent}(\$ag_id)) = \{\sigma_{IdAgent=\$ag_id} \text{AGENT}, \sigma_{IdAgent=\$ag_id} \text{AGENDA}, \\ \text{VISIT}, \text{SALES}, \text{RENT}, \text{OWNER}, \text{CUSTOMER}, \text{ESTATE}, \text{PICTURE}\}$$

Consider now the context of an agent $\$ag_id$ specified by the configuration $C = \langle \text{role} : \text{agent}(\$ag_id), \text{category} : \text{residential} \rangle$. The view obtained by applying the Double Intersection operator between $\mathcal{R}el(\text{agent}(\$ag_id))$ and $\mathcal{R}el(\text{residential})$, whose partial view is that reported in Example 1, is the following:

$$\mathcal{R}el(C) = \mathcal{R}el(\text{agent}(\$ag_id)) \pitchfork \mathcal{R}el(\text{residential}) \\ = \{\sigma_{Category=\text{Residential}} \text{ESTATE}, \text{OWNER} \times (\sigma_{Category=\text{Residential}} \text{ESTATE}), \\ \text{VISIT} \times (\sigma_{Category=\text{Residential}} \text{ESTATE}), \text{SALES} \times (\sigma_{Category=\text{Residential}} \text{ESTATE}), \\ \text{RENT} \times (\sigma_{Category=\text{Residential}} \text{ESTATE}), \text{PICTURE} \times (\sigma_{Category=\text{Residential}} \text{ESTATE}), \\ (\sigma_{IdAgent=\$ag_id} (\text{AGENDA})) \times (\sigma_{Category=\text{Residential}} \text{ESTATE})\} \blacklozenge$$

For the sake of clarity, in the examples we have used configurations with two context elements only, yet the *commutative* and *associative* properties of the introduced Double Union and Double Intersection operators allow their application to configurations C with any number of context elements.

These logical operators have been defined to support our methodology for view composition, a problem deeply formalized from a theoretical point of view in [11]; nevertheless they are general-purpose integration operators, appropriate for any schema and instance integration over relational schemata. Within our context-driven methodology, a few properties useful for the final view composition phase can be derived.

View Composition Properties

The mapping and the compositional operators used to obtain a context-dependent view allows us to prove the following properties. The next two theorems show how, given the view for a configuration C related to a context described by k context elements (i.e. $C = \langle V_1, \dots, V_k \rangle$), we can obtain the view for a configuration with an additional context element w.r.t. C (i.e. $C' = \langle V_1, \dots, V_k, V_{k+1} \rangle$), by simply composing (by Double Union or Double Intersection) the partial view of V_{k+1} with $\mathcal{R}el(C)$.

Property 2. *Let $\mathcal{R}el(\langle V_1, \dots, V_k \rangle)$ be the partial view defined for the configuration $C = \langle V_1, \dots, V_k \rangle$ w.r.t. the maximal area policy. Then:*

$$\mathcal{R}el(\langle V_1, \dots, V_k, V_{k+1} \rangle) = \mathcal{R}el(\langle V_1, \dots, V_k \rangle) \pitchfork \mathcal{R}el(V_{k+1})$$

Proof. Starting from the fact that \pitchfork is a commutative and associative operator, by applying its definition to a set of $k+1$ operands we obtain:

$$\begin{aligned} \mathcal{R}el(\langle V_1, \dots, V_k, V_{k+1} \rangle) &= \mathcal{R}el(V_1) \pitchfork \dots \pitchfork \mathcal{R}el(V_k) \pitchfork \mathcal{R}el(V_{k+1}) \\ &= (\mathcal{R}el(V_1) \pitchfork \dots \pitchfork \mathcal{R}el(V_k)) \pitchfork \mathcal{R}el(V_{k+1}) \\ &= \mathcal{R}el(\langle V_1, \dots, V_k \rangle) \pitchfork \mathcal{R}el(V_{k+1}) \quad \square \end{aligned}$$

Property 3. *Let $\mathcal{R}el(\langle V_1, \dots, V_k \rangle)$ be the partial view defined for the configuration $C = \langle V_1, \dots, V_k \rangle$ w.r.t. the minimal area policy. Then:*

$$\mathcal{R}el(\langle V_1, \dots, V_k, V_{k+1} \rangle) = \mathcal{R}el(\langle V_1, \dots, V_k \rangle) \cup \mathcal{R}el(V_{k+1})$$

Proof. The proof is the same of Theorem 2 where the operator \pitchfork is replaced with \cup . \square

The next theorem extends the containment relationships between partial views of CDT context elements (see Assumption 1) to views.

Property 4. *Let $C = \langle V_1, \dots, V, \dots, V_k \rangle$ be a configuration and V, W two context elements such that $V \prec W$. Then:*

$$\mathcal{R}el(\langle V_1, \dots, V, \dots, V_k \rangle) \supseteq \mathcal{R}el(\langle V_1, \dots, W, \dots, V_k \rangle)$$

Proof. This result is independent of the operator used to compose partial views. Let us now consider the case of composition by means of the Double Intersection operator. According with Assumption 1, since $V \prec W$, then $\mathcal{R}el(V) \supseteq \mathcal{R}el(W)$. Hence:

$$\begin{aligned} \mathcal{R}el(\langle V_1, \dots, V, \dots, V_k \rangle) &= \mathcal{R}el(V_1) \pitchfork \dots \pitchfork \mathcal{R}el(V) \pitchfork \mathcal{R}el(V_k) \\ &\supseteq \mathcal{R}el(V_1) \pitchfork \dots \pitchfork \mathcal{R}el(W) \pitchfork \mathcal{R}el(V_k) \\ &= \mathcal{R}el(\langle V_1, \dots, W, \dots, V_k \rangle) \quad \square \end{aligned}$$

The view associated with a specific context can include or not some portions of the global database on the basis of (i) the width of the partial views specified during the tailoring process for each context element and (ii) the operator used to combine partial views.

The different policies that can be adopted to identify the relevant area associated with a context element impose the use of different operators for combining such partial views to obtain the final ones. These operators can be applied for two purposes: (i) to derive the view associated with a configuration, starting from the partial views, and (ii) to define the partial view associated with non-leaf nodes, as discussed for the bottom-up navigation of the Context Dimension Tree during the relevant area assignment phase.

4 Closing Remarks and Future Work

The approach presented in this paper aims at providing a semi-automatic support to view definition in a relational database scenario, taking into account various possible users, situations and interests that lead to the selection of different portions of the global schema and data. To this end, we have formally introduced a group of logical operators which allow the combination of partial views incrementally derived on the global schema.

A prototype tool has been developed to support the designer in all the phases of the methodology, from the design of the Context Dimension Tree and the *Relevant Area Assignment* to the automatic *View Composition* phase. The designer can then review the obtained views associated with the significant configurations corresponding to the application contexts, and eventually introduce modifications to fulfill her/his needs.

These operators and their application have been here presented within the relational framework; current research is focused on the extension of this methodology to other scenarios, characterized by semi-structured sources, to cope with nowadays variety of information representations.

Acknowledgements. The authors wish to thank Prof. Letizia Tanca for the helpful discussions.

References

1. Abiteboul, S.: On Views and XML. In: Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, pp. 1–9. ACM Press, New York (1999)
2. Bolchini, C., Curino, C., Quintarelli, E., Schreiber, F.A., Tanca, L.: Context information for knowledge reshaping. *Int. Journal on Web Engineering and Technology* (to appear 2007)
3. Bolchini, C., Curino, C.A., Orsi, G., Quintarelli, E., Schreiber, F.A., Tanca, L.: CADD: a tool for context modeling and data tailoring. In: Proc. IEEE/ACM Int. Conf. on Mobile Data Management - Demo Session, May 7-11, ACM Press, New York (2007)
4. Bolchini, C., Quintarelli, E.: Filtering mobile data by means of context: a methodology. In: Meersman, R., Tari, Z., Herrero, P. (eds.) *OTM 2006 Workshops*. LNCS, vol. 4278, pp. 1986–1995. Springer, Heidelberg (2006)
5. Bolchini, C., Quintarelli, E.: Filtering mobile data by means of context: a methodology. In: Proc. 2nd Int. Workshop on Context Representation and Reasoning, pp. 13–18 (2006)
6. Bolchini, C., Schreiber, F.A., Tanca, L.: A methodology for very small database design. *Information Systems* 32(1), 61–82 (2007)
7. Cluet, S., Veltri, P., Vodislav, D.: Views in a large scale of XML repository. In: Proc. 27th Int. Conf. on Very Large Data Bases, pp. 271–289 (2001)
8. Elmasri, R., Navathe, S.: *Fundamentals of database systems*, 4th edn. Pearson/Addison Wesley (2004)
9. Ghidini, C., Giunchiglia, F.: Local Models Semantics, or contextual reasoning=locality+compatibility. *Artificial Intelligence* 127(2), 221–259 (2001)
10. Liefke, H., Davidson, S.B.: View maintenance for hierarchical semistructured data. In: Kambayashi, Y., Mohania, M.K., Tjoa, A.M. (eds.) *DaWaK 2000*. LNCS, vol. 1874, pp. 114–125. Springer, Heidelberg (2000)
11. Madhavan, J., Halevy, A.Y.: Composing mappings among data sources. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) *Databases, Information Systems, and Peer-to-Peer Computing*. LNCS, vol. 2944, pp. 572–583. Springer, Heidelberg (2004)

12. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview, W3C Recommendation (2004)
13. Rajugan, R., Chang, E., Dillon, T.S.: Ontology views: A theoretical perspective. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1814–1824. Springer, Heidelberg (2006)
14. Rajugan, R., Dillon, T.S., Chang, E., Feng, L.: A Layered View Model for XML Repositories and XML Data Warehouses. In: Proc. IEEE 5th Int. Conf. on Computer and Information Technology, pp. 206–215. IEEE Computer Society Press, Los Alamitos (2005)
15. Roussos, Y., Stavarakas, Y., Pavlaki, V.: Towards a context-aware relational model. In: Proc. Context Representation and Reasoning - CRR'05, pp. 7.1–7.12 (2005)
16. Stefanidis, K., Pitoura, E., Vassiliadis, P.: Modeling and storing context-aware preferences. In: Manolopoulos, Y., Pokorný, J., Sellis, T. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 124–140. Springer, Heidelberg (2006)
17. Volz, R., Oberle, D., Studer, R.: Implementing views for light-weight web ontologies. In: Proc. IEEE 7th Int. Database Engineering and Applications Symp., pp. 160–169. IEEE Computer Society Press, Los Alamitos (2003)
18. Volz, R., Oberle, D., Studer, R., Staab, S.: Views for light-weight web ontologies. In: Proc. ACM Symp. on Applied Computing, pp. 1168–1173. ACM Press, New York (2003)
19. Wouters, C., Rajugan, R., Dillon, T.S., Rahayu, J.W.R.: Ontology extraction using views for semantic web. In: Web Semantics and Ontology, pp. 1–40. Idea Group Pub., USA (2005)
20. Zhuge, Y., Garcia-Molina, H.: Graph structured views and their incremental maintenance. In: Proc. IEEE 14th Int. Conf. on Data Engineering, pp. 116–125. IEEE Computer Society Press, Los Alamitos (1998)