CARVE: Context-aware automatic view definition over relational databases

Cristiana Bolchini, Elisa Quintarelli *, Letizia Tanca

Dipartimento di Elettronica e Informazione—Politecnico di Milano, Piazza Leonardo da Vinci, 32—20133 Milano, Italy

ARTICLE INFO

Article history:
Received 12 January 2011
Received in revised form 2 April 2012
Accepted 4 May 2012
Recommended by: J. Van den Bussche
Available online 12 May 2012

Keywords:
Context-awareness
Database design
External views

ABSTRACT

Classical database design strategies, based on the a priori definition of application views to be finally integrated in the design of the global databases, are not appropriate for the design needs of modern, highly dynamic information systems. This paper presents CARVE (Context-Aware Relational View dEfinition), a methodology for context-aware view definition, well-suited for the design of modern, dynamic applications that, in different environments and situations, need to access different portions of data. The methodological approach includes a context-design phase, followed by a phase when each of the possible application contexts is automatically associated with its relevant part of information (context-aware view). Accordingly, CARVE is based on a context model, on guidelines to define partial views related to components of the context, and on a set of operators used to perform partial-view composition to derive the context-aware views. The paper leverages on previously presented preliminary results and introduces and formalizes the overall methodology and its steps, providing a revised and assessed approach. The results of the application of the approach to a set of case studies are reported, together with a careful evaluation.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Modern information systems are more and more dynamic in many respects: on one hand, frequent integrations with other systems cause the need for numerous modifications of the application programs; on the other hand, there is a ubiquitous hunger for information, which causes users to expect, anywhere at any time, to obtain quick access to the data they need, in their current situation, that is, appropriately tailored with respect to their context.

According to Dey [1], we refer to context as any information that can be used to characterize the situation of an entity, where an entity is anything considered as relevant for the interaction between a user and an application. Context has often a significant impact on the way humans (or machines) act, and on how they interpret things; it has indeed been recognized by Brézillon and Abu-Hakima [2] that “knowledge has a contextual component”, and that this component may be of use to “extract and present the relevant chunks of knowledge”, thus allowing for information focusing and reduction.

Classical database design strategies [3], based on the a priori definition of static application views to be finally integrated in the design of the global databases, are doomed to become inappropriate due to the need to accommodate the high dynamicity of new information systems. By contrast, the need arises to conceive fresh user views whenever the occurred changes require it, thus a flexible methodology supporting (possibly a posteriori) definition of the appropriate views over databases is in order, along with the means for their agile adaptation during the system’s life so that a fresh view is delivered to the user or to the application at each change of situation. This data design scenario calls for a new kind of abstraction mechanism, called viewpoint abstraction [4], which becomes fundamental for the design of modern information bases. Viewpoint abstraction aims at
considering the various contexts in which the data are used by the different stakeholders, to derive the appropriate context-aware views from the information needs of each stakeholder in each specific context. Moreover, we cannot think of changing the application code each time the context varies; it is thus advisable that a mechanism exists allowing applications to access the data in a transparent way, with no code changes in the points related to database access. Note that this last aspect acquires even more importance when dealing with small, portable devices with limited storage, computational and power resources: in this case, at each change of context the related context-aware view should be materialized and delivered to the device, so that the user owns a local copy of the context-relevant data.

We thus advocate a new view-design methodology where, instead of being fixed once and for all for each given user or application, the views automatically adapt to changes in the situation or context. The methodology comprises a sequence of steps for defining and computing these contextual views; note that the schema the views are defined over might be the logical schema of a centralized database or the global schema of a multi-database; this point is irrelevant as long as a global schema exists over which context-aware, dynamic external views – as in the ANSI-SPARC model [5,6] – can be defined.

Differently from other approaches to context-aware system design [7], we believe that the context in which information is managed is orthogonal with respect to what we might call “object information”, and that as such it should be treated. Consequently we recommend the use of a context model which is completely independent of the information space, and whose relationship with it is clearly stated. Accordingly, the main contribution of this paper is the CARVE methodology, where the context is initially modeled independently of the data, and then used as a guide to tailor the external views over the database. With CARVE, either at database design from scratch or when re-designing the user views over legacy database(s), the database designer is guided, on one hand, in the conception of the possible contexts pertaining the reality of interest and, on the other hand, in assigning appropriate views to each of these different contexts. We claim that the complexity of such operations is greatly reduced by the application of the steps proposed by CARVE, and by the support provided by the associated CADEFrame tool. Moreover, CARVE favors dynamism, in that it permits easy run-time revision of the context-aware views.

We build on some research already published in other works by the same authors [8–10]; the original contributions we present in this paper are:

- the overall organization of the CARVE methodology, presented step by step;
- an improved version of the context model we introduced in [10]. In this new definition, we add new elements useful to model and express the aspects necessary to specify the characterization of final users;
- refinement and correction of the view composition operators [8] and of the strategies to identify the database portion interesting for a given context; their previous versions were tested against the case studies presented in Section 7, and they revealed themselves inadequate to capture all the possible relationships that a designer might be wanting to establish between a context and the associated data. This also originated the introduction of a new strategy for view composition;
- the recognition and study of mutual relationships between context constraints and usage of the operators;
- proofs of new useful properties of the operators, and finally
- the prototype tool we developed to support CARVE.

The rest of this paper is organized as follows. Section 2 presents the motivations behind this work and the rationale leading to the methodology we have developed; it also introduces a running example used in the rest of the paper. Section 3 discusses the related work available in the literature, with reference to the main contributions of the paper. Section 4 is preliminary to the description of the CARVE methodology because it presents the relational set-oriented operators we have defined for view composition. The context model and the CARVE methodology are presented in Sections 5 and 6, respectively. In particular, Section 6.6 focuses the attention on the impact of the context-model constraints on view definition. Section 7 describes the prototype tool supporting our approach, while Section 8 proposes an evaluation of CARVE and Section 9 closes the paper.

2. Motivations and rationale

We propose to use the context abstraction as a guide to select the portions of data which are deemed interesting in the various situations of the information system life. In this paper, we refer to the relational data model, thus these interesting data portions correspond to one or more relational views over a global schema, in general to a set of views; in the following, by the general word “view” we will actually mean a set of (virtual) relations.

While the computer science community has initially perceived the context simply as a matter of user time and location, in the last few years the context has been analyzed as a set of variables, also called dimensions, whose values may be of interest for a human or an application because they influence its actions. In CARVE we propose the Context Dimension Tree (CDT) for modeling at design time all the possible contexts the user will run the application in; in the Context Dimension Tree (thoroughly presented in Section 5), the various aspects of context, its elements, such as the user’s location or his/her current role and interests, are represented as nodes of the tree, so that each context is a set of nodes.

Once all the possible contexts have been generated, a context-aware view over the global schema will be assigned to each of them. At run-time, when a context is active, the user application queries will be seamlessly applied over the context-aware view associated with it, without further pre-processing.

The design-time activity of CARVE is depicted in Fig. 1. In the first phase, “Application Design”, besides designing data and application functions as usual, the designer identifies the elements characterizing the application contexts
and models them by means of the CDT. Note that, while context influences knowledge, it is also a part of it, thus deciding which information constitutes the context and which constitutes the “object data” is almost a philosophical problem. A simplistic, “Occam’s razor” approach is to say that this is a design decision, and actually it is. However, we feel that a short discussion is in order, since designers should at least be provided with some guidelines to take this choice. In Section 8, while evaluating our approach, we introduce a small discussion on this topic.

In the second phase, “Partial View Definition”, the designer must specify, for each context element, the associated portion of data (partial view over the global database), to determine the significant information related to it (as defined in Section 6.1).

In the “View Composition” Phase all views over the database (each associated with a possible envisioned context) are automatically generated (Section 6.2). Each of these views represents the pertinent, context-aware data to be made available to the user in a specific context; these data will be queried later, at run time, when that context becomes active.

Thus CARVE supports semi-automatic assignment of different views over the global schema to the different contexts. This assignment cannot be entirely manual for two reasons: (a) the number of possible contexts may be very high, (b) new “context dimensions”, i.e., analysis perspectives of context, might arise at run-time: in this case, the association of data with contexts must be revised and automatic mechanisms to support evolution have to be envisioned [11].

Since the “naive” context-view association strategy, based on a per-context approach, is inappropriate, we propose here a per-element strategy which takes into account the aspects modeled in the context independently one of the other, and identifies, for each corresponding node of the tree (context element), a partial view. The final view associated with a context is computed by properly combining the partial views associated with the nodes constituting that specific context. In this paper we refine a preliminary proposal of a set of operators adopted to combine partial views to derive the context-aware ones.

In the per-element strategy, the designer only has to specify one partial view for each context element, whereas the computation of the final views associated with each context can be automatically performed by a tool. As a result, the required effort is significantly lower and there is the possibility to adapt the views incrementally, when new context perspectives arise. On the other hand, it might happen that the final views obtained in this way be not satisfactory; for this reason we discuss in Section 6 the possibility that CDT dimensions be not completely orthogonal and propose solutions to this problem. Eventually, we envision a last phase, where the designer may review the final views s/he wants to better tailor, by specifically adding, excluding or further aggregating information pieces.

**Fig. 1.** The CARVE methodology.
As discussed later on, the operators for automatic view combination need to satisfy two fundamental requirements:

1. **Query resilience**: the application of our operators should leave the input relational database schema “as intact as possible”, without defining new relations. This requirement is important with respect to the development of context-aware applications: as long as the tailored portion of data refers to the same relational schema (possibly where relations do not have all attributes and tuples), the application designer is relieved from the burden of designing different application queries for the different contexts, relying on the fact that queries to retrieve data are applicable across the different schemas in the different resulting contexts.

2. **Set orientation**: since, as noted above, the partial view associated with each context element actually defines a set of virtual relations, the new operators should work on relation sets.

The set-oriented operators, based on the classical relational algebra ones, are presented in Section 4, while CARVE is presented in detail in Section 6.

### 2.1. Running example

As an example for the application of the proposed data tailoring methodology, which also encompasses the definition of constraints to prevent the definition of unsuitable contexts, consider the scenario of a group of independent restaurants joining forces to promote themselves by offering their services through the “Pick-up Your Lunch” (PYL) corporation. These restaurants, together with the traditional activity, offer on-line ordering for either pick-up or delivery, allowing customers to put together their favorite meal also by taking separate dishes from different restaurants. The delivery service is provided by the corporation, joined by a group of taxi companies, who deliver the assembled meals. Pick-up service is also offered, allowing customers to order their meal, in the same flexible fashion, and to pick it up on their way to the park or home, at one of the numerous pick-up sites. Finally, a web site is used to promote the business, where prospective customers can browse and search the restaurants and menus. Registered users, i.e., customers, can download on their mobile smartphone a small application to perform the basic operations and interact with the PYL server, in order to retrieve up-to-date information on the restaurants, menus and dishes available with respect to the selected (or perceived) location and time. More precisely, the main operations the application supports are the following ones: (i) browse the menus and browse/find the restaurants, (ii) order a meal, either to be picked up or delivered, (iii) review past orders, (iv) make a reservation.

The application works with local data, and can – whenever a connection is available – update such data by downloading the portion of the information available on the PYL server, corresponding to the user’s active context.

The relational schema constituting the PYL Server database, storing all information, is reported in Fig. 2.

In this scenario it is possible to identify four types of users, each one interested in a subset of the entire

---

**Fig. 2.** The PYL database schema.
data: customers who are registered users, restaurant owners, taxi companies interested in open orders for delivery, and guests, who navigate the web site of the PYL company, without being registered users. All users but the last ones are precisely identified since they will access their own records, whereas guest users are not individually identified.

As stated, the information concerning restaurants or pick-up places may be related to the user’s location (the customer) which may be explicitly set by her/him or “perceived” by the smartphone/pda (through the GPS, for instance). As a result, information selection can be performed with respect to the location by stating the zone of interest or a range of distance with respect to a given position.

Another perspective to limit the amount of information the user is interested in refers to the interface of the application used for accessing data; when a smartphone or PDA is used, only minimal information should be exchanged; on the other hand, when using a web browser on a normal Personal Computer (e.g., via web), the information can be complete.

According to the CARVE methodology all these aspects are taken into account at design time, while foreseeing the final application scenario. At run time, then, the user will access through a context-aware application the available information, which, based on the user’s current context (partially autonomously perceived – as in the case of time and location – and partially explicitly declared) will be tailored and made available.

The PYL customer is the main actor of the application scenario. Let us imagine a person on her way to work, who is planning to pick up her lunch at noon and eaten out at the park; she is interested in today’s Japanese menus, to be available for pick-up around her office. She accesses through her smartphone the PYL services, specifying her preferences related to the lunch, the Japanese cuisine and the pickup option. As a result, she will receive all information matching those tailoring criteria, deriving from the execution of the views defined at design time, actualized with the active context-dependent values.

As another example, a worker of the taxi company, who is located in a certain zone of the city, is interested in the open delivery orders within a certain distance from his/her actual location, needing to know where to gather the food to be delivered and the final destination address.

It is worth noting that the exploitation of the user’s context aims at reducing the amount of information retrieved and locally stored, to improve performance and reduce (connection) costs. Indeed, the entire database is particularly rich, listing hundreds of restaurants and thousands of ingredients, therefore the possibility to tailor the amount of interesting data is significant.

3. Related work

Works that deal with similar problems as ours belong to two different categories. The first category includes the plethora of recent research on context definition [12–14] and context-related operators [15,16]. Here context is elevated to the role of first-class citizen; however (sometimes purposefully), there is no clear separation between the (meta-level) concept of context and the (object-level) database.

Recent studies make such an attempt [17,18]. Starting from similar considerations as ours, Vieira et al. [17] propose a general approach to deal with context modeling through a clear separation between context concepts (called contextual elements) and domain/application concepts. As in our case, the needs which motivate this research are: (i) formalization of the contextual concepts that will guide the context management; (ii) identification of the contextual concepts relevant for a target domain; (iii) instantiation of the contextual elements to the application usage. However, Vieira et al. [17] as well as Henrickson et al. [19] base context representation on ontologies, whereas our model [8,9], the CDT, is independent of the representation formalism: we are actually experimenting different formalisms [20,21], adopted according to the specific task the model is used for.1 Because of this independence of the representation formalism, in application scenarios where reasoning capabilities are needed, these can be introduced by representing the CDT by means of an ontology or by some logic-programming language [22]. Other strong points are that the CDT encompasses the inherently hierarchical nature of contexts and focuses on the possibility to represent mutual relationships between different contexts, and constraints on their instantiation.

Roussos et al. [18] recognize the need for an orthogonal context model; indeed, context is considered as a first class citizen, and operations on contexts and context-aware relations are defined, by extending the relational model and (traditional) relational algebra. However, their work proposes an extension of relational algebra operators to a three-dimensioned, context-aware relational model, while we define operators which act on sets of relations, are quite independent of the notion of context, and may be used either (as we propose) within a context-oriented derivation of user views or as stand-alone operators (e.g. for data integration), thus accomplishing full orthogonality. Stavrakas et al. [23] present a query language that supports context-driven queries and show the benefits of considering context as first class citizen.

Though all these works recognize the orthogonality of context, we believe that the main difference with our work is that we do not mix context modeling and representation with the language(s) and models that define the data. This allows our approach to be applied to any already existing system without changing the representation and query formalisms.

In the second category of related work we cite research on view integration or composition within the area of data integration. For example, Motro [24], within the typical approach to data-integration called GAV [25], presents a formal definition of schema integration operations, for defining a virtual global schema (obtained from the integration of multiple databases) as a superview. Such superview is obtained by iteratively building views over the data sources.

---

1 For example we adopt a DL-based representation of the CDT for context-aware data integration.
and combining them into more complex views by means of appropriate operators. During this process, also a set of global constraints is defined. Because of the fact that the final goal of CARVE and the one presented by Motro differ, the defined operators are rather different; nevertheless, the idea of building superviews is similar.

Other notable operators for view composition are proposed for schema merging [26,27]; a merge operator is defined and used with different flavors, based on the integration needs [28,29]. In all these researches, the defined operators work on individual relations rather than on sets of them, as the ones that are more appropriate here. In some approaches, both intensional and extensional integrations are performed, by defining a unified schema including all sources (intensional), a set of operators to perform the integration while guaranteeing semantic equivalence between the resulting schema and the initial ones, and a set of transformations of the data, to populate the resulting schema (in the case of extensional integration like in the data warehousing case).

This perspective concentrates on achieving semantic equivalence between schemata [30], thus focusing on the correspondence of elements of different schemata, whatever format they have (relational and conceptual models [25,31–33], graph-based [34] or generic [35] models). The aim of these approaches is to solve the integration problem in general, while our objective is to find operators that act on sets of relations and have the limited expressivity required for our work.

4. View composition operators

In this section we introduce the operators, based on relational algebra, that will be used to compose so-called partial views and produce new, context-aware ones. Beside the examples included in this section, more concrete instances of the application of the operators are reported in Section 6.

The process of associating views with contexts requires the following inductive definition of the notion of a tailoring expression.

In the following let \( D \) be a database schema, and \( R \) a generic relation schema of \( D \). We assume that each relation schema \( R \in D \) has a designated primary key \( K(R) \), and denote with \( \text{Att}(R) \) the set of attributes of \( R \). Given a relational algebra expression \( rae \), we indicate with \( rae(I) \) the relation obtained by applying \( rae \) to an instance \( I \) of \( D \), and with \( \text{Att}(rae) \) the result attributes of \( rae \), i.e., the attributes of any relation obtained by applying \( rae \) to whatever instance \( I \) of \( D \).

**Definition 1.** For each relation \( R \), let \( X \subseteq K(R) \) be a subset of attributes of \( R \) that contains its key attributes, \( \theta \) a boolean condition on \( \text{Att}(R) \), and \( rae \) a relational algebra expression on \( D \). A tailoring expression \( e^R \) over \( R \) is inductively defined as follows:

(a) \( e^R = PI_X\sigma_\theta(R \bowtie rae) \) is a tailoring expression;
(b) if \( e_1^R \) and \( e_2^R \) are tailoring expressions over \( R \) with \( \text{Att}(e_1^R) = \text{Att}(e_2^R) \), then \( e_1^R \cup e_2^R \) is a tailoring expression over \( R \);
(c) if \( e_1^R \) and \( e_2^R \) are tailoring expressions over \( R \) with \( \text{Att}(e_1^R) = \text{Att}(e_2^R) \), then \( e_1^R \cap e_2^R \) is a tailoring expression over \( R \).

Note that the intersection operator is not mentioned in **Definition 1**, because it can be derived from the union and difference operators; however, to make the expressions more intuitive, we will use it in the examples.

It is easy to see that:

**Property.** Each tailoring expression \( e^R \) over \( R \) can be equivalently expressed as \( \Pi_X \exp \), where \( \exp \) is obtained from \( e^R \) by removing all the projection operators and \( X \subseteq \text{Att}(R) \).

Given a tailoring expression \( e^R \) over \( R \), is called origin of \( e^R \) (also denoted as \( O(e^R) \)).

Examples of tailoring expressions are:

**DELIVERY \( \bowtie \) \( \sigma_{\text{CustomerID} = \text{SciID}} \)**

**ORDERS**

**NORDER**

**ORDERDETAIL**

**CUSTOMERS**

The partial views mentioned in Section 2 are realized as sets of tailoring expressions, which must be “combined” (or composed) to derive other such sets to be associated with a given context. For this reason, the schema of a tailoring expression is a subschema of that of its origin, because we want that each relational schema contained in the context-aware views be “as intact as possible” with respect to the initial schema, as anticipated in Section 2. Consequently, tailoring expressions are less expressive than relational algebra expressions, for instance they do not include the possibility to use the join operator, but only the semi-join.

In the following, we present the algebraic operators which are the basis for the composition of partial views (“View Composition Phase” of Fig. 1). Note that these operators find useful application also independently of the CARVE methodology: for instance, as already noted when discussing related work, in GAV-based data integration [25] a simple global schema over the data-sources may be defined by means of our operators. In this case, the global schema would only contain relations whose schemata are subschemas of some relation present in at least one of the data sources, because of the limitation of the expressive power of the tailoring expressions. The advantage of such simplistic approach is that any application designed for one of the data sources would be able to access the global schema without any code modification, that is, the query resilience property is guaranteed.

The re-join operator, introduced next, forms the basis for the set-based operators that follow.

4.1. Re-join

Among the operators we define, re-join (\( RJ \)) is the only one which takes as input a pair of relational algebra expressions and not a pair of sets thereof, and is instrumental for the definition of the other operators; the re-join is an extension of the outer-join operator, and can be also serviceable in normal database operations when
more information is available than in the classical situation where the outer-join is used.

A well-known extension of the join operator is the outer-join (OJ), defined in [3] that, unlike a regular join (or inner-join), keeps both matched and unmatched tuples from the two input relations \( R_1 \) and \( R_2 \). For the unmatched tuples of \( R_1 \), null padding is applied for attributes of \( R_2 \) not present in the schema of \( R_1 \), and vice versa. As an example, suppose two tailoring expressions \( e_i^R \) and \( e_2^R \) have been defined over \( R \). Fig. 3(a)–(c) visually shows the initial relation, the inner-join and the outer-join between them, respectively, highlighted by the bold rectangle.

The re-join operator, defined only between two tailoring expressions over the same origin \( R \), is a variation of the outer-join that can be used when more information is known. The (extensional) effect of the operator applied to \( e_1^R \) and \( e_2^R \) is shown in Fig. 3(d), where it is visually compared with the other discussed joins. We need the re-join when combining partial views, which are formed by tailoring expressions that will only be materialized when the current context-aware view is computed and which are often derived from the same original relation. Actually when an outer-join is performed between two tailoring expressions \( e_1^R \) and \( e_2^R \), the system knows more information (i.e. the intensional definition of the expressions and not only the extensional views as two sets of tuples) than when performing a regular outer-join between any two tables; thus, for each attribute that belongs to \( \text{Att}(e_1^R) \) but not to \( \text{Att}(e_2^R) \), instead of applying null padding, the re-join includes the known values of \( R \)'s attributes.

**Definition 2.** Let \( R \) be a relational schema, \( X \) and \( Y \) sets of attributes of \( R \), and \( \theta_1 \) and \( \theta_2 \) conditions over \( \text{Att}(R) \). Given two tailoring expressions over \( R \), respectively, \( e_1^R = \Pi_X \exp_1 \) and \( e_2^R = \Pi_Y \exp_2 \), their re-join (RJ) is defined as follows:

\[
e_i^R \cap e_2^R = \Pi_{X \cup Y}(\exp_1 \cup \exp_2)
\]

Note that, in the particular case that \( e_i^R = \Pi_X \sigma_{\theta_i}(R) \) and \( e_2^R = \Pi_Y \sigma_{\theta_2}(R) \), we have \( e_i^R \cap e_2^R = \Pi_{X \cup Y} \sigma_{\theta_1 \cup \theta_2}(R) \). Note also that, if \( e_i^R \) and \( e_2^R \) are two tailoring expressions over \( R \), then \( e_i^R \cap e_2^R \) is also a tailoring expression over \( R \).

In the following subsections we define the three fundamental operators that apply to sets of tailoring expressions and that are proposed for partial view composition, along with some simple examples. Other, more complex examples will be presented in Section 6, where their concrete use is introduced in relation to partial view composition.

**4.2. Double-intersection**

Let \( A \) and \( B \) be two sets of tailoring expressions. For each pair of tailoring expressions \( e_i^A \in A \) and \( e_j^B \in B \), let \( X = \text{Att}(e_i^A) \cap \text{Att}(e_j^B) \). As defined by the algorithm of Fig. 4, the double-intersection operator \( A \cap B \), between \( A \) and \( B \) applies the classical intersection operator \( \cap \) to pairs of expressions over the same origin. The projection over the intersection \( X \) of the schemas is performed to reduce the input tailoring expressions to a common schema.

A preliminary, less general definition of this operator has been given in [8].

**Example 1.** Given two relations \( R_1(K_1, A_1, A_2) \), \( R_2(K_2, B_1, B_2, B_3) \), and two sets of tailoring expressions \( A = \{ K_1, \pi_{K_1} y_1 \land x R_1 \} \) and \( B = \{ K_2, \pi_{K_2} y_2 \land y R_2 \} \), then \( A \cap B = \{ K_1, K_2, \pi_{K_1} y_1 = \pi_{K_2} y_2 \} \).

It can be easily shown that double-intersection is commutative and associative [8]. Moreover, note also that, if \( A \subseteq A' \), then \( A \cap B \subseteq A' \cap B \) (with \( A \), \( A' \), \( B \) three sets of tailoring expressions); that is, double-intersection is monotonic w.r.t. set inclusion.

![Fig. 3. The join operators extensional effect (highlighted by the bold border rectangle): (a) the initial expressions, (b) their inner-join, (c) their outer-join and (d) their re-join.](image)

![Fig. 4. The double-intersection algorithm.](image)
Independently of the CARVE methodology, double-intersection can be applied to build a unified view of two (or more) datasources when we are only interested in the information shared by (some parts of) the datasources. This may be the case, for example, of portions of data coming from different administrations, which are considered trustable only if present in at least two data sources.

4.3. Double-union

Let $A$ and $B$ be two sets of tailoring expressions. The double-union operator $A \cup B$ of $A$ and $B$ returns (1) all expressions $e_A^p$ in $A$, whose origin is not present as origin in any expression of $B$, (2) all expressions $e_B^p$ in $B$, whose origin is not present as origin in any expression of $A$, and (3) for each pair of expressions $e_A^p \in A$ and $e_B^p \in B$, the re-join of $e_A^p$ and $e_B^p$.

The algorithm for computing the double-union operator is reported in Fig. 5. Note that, as double-intersection, also double-union is commutative and associative [8]. Monotonicity w.r.t. set inclusion holds also for double-union, i.e. $A \subseteq A' \Rightarrow A \cup B \subseteq A' \cup B$, for any $A,A',B$.

Example 2. Given two relations $R_1(K_1, A_1, A_2), R_2(K_2, B_1, B_2, B_3)$, and two sets of tailoring expressions $A = \{R_1, \pi_{K_1,B_1}, \sigma_{B_1 = aR_2}\}$ and $B = \{\pi_{K_2,B_2}, \sigma_{B_2 = y(R_2)}\}$, we have $A \cup B = \{R_1, \pi_{K_2,B_1}, B_1, B_2, B_3 \mid x.R_1 = y.R_2\}$. □

Double-union is also an operator of general interest. Indeed, it is useful to construct a unified view over data sources where we want to take everything. Here the presence of the re-join operator also guarantees semantic coherence when we are reconstructing relations that had been previously partitioned either vertically or horizontally [5].

4.4. Double-minus

Let $A$ and $B$ be two sets of tailoring expressions. The double-minus operator $A \setminus B$, between $A$ and $B$ returns a set of relations composed by (1) the expressions belonging to $A$ such that there is no expression in $B$ with the same origin and (2) for each pair of expressions $e_A^p$ and $e_B^p$ in $A$ and $B$, respectively, the expression composed by the tuples of $e_B^p$ that are in the relational-algebra difference applied on the common sub-schema of $e_A^p$ and $e_B^p$ (i.e. $e_B^p := (\pi_K(e_A^p), \pi_K(e_B^p))$, with $K$ the key attributes of $R_i$), and for these tuples only the attributes $\operatorname{Att}(e_B^p) - \operatorname{Att}(e_A^p)$ plus key attributes $(\pi_{K, \operatorname{Att}(e_B^p) - \operatorname{Att}(e_A^p)}(e_B^p := (\pi_K(e_A^p), \pi_K(e_B^p))))$.

that, for each expression in the result, key attributes are kept in its schema. The operator is not commutative. See Fig. 6 for a procedural definition of the double-minus.

Example 3. Given two relations $R_1(K_1,A_1,A_2), R_2(K_2, B_1,B_2, B_3)$, and two sets of tailoring expressions $A = \{R_1, R_2\}$ and $B = \{\pi_{K_2,B_1}, \sigma_{B_1 = a(R_2)}\}$, then $A \setminus B = \{R_1, \pi_{K_1,B_1}, B_1, B_2, B_3 \mid x.R_1 = y.R_2\}$. Indeed, $\pi_{K_1,B_1}(R_1) \subset \pi_{K_1,B_1}(R_2)$.

Apart from the cases we illustrate later in this paper, double-minus can be used whenever, in a unified view over data sources, we want to hide some portions of data and schemas, for instance for privacy preservation.

Fig. 7 contains (a) a visual representation of two sets of tailoring expressions, namely $A$ and $B$, each showing the origin relations too, together with the result of applying (b) double-intersection, (c) double-union and (d) double-minus.

Note that these operators can obviously be realized in relational algebra. However, they are useful as each of them defines the synthetic application of the same operator (intersection, union or difference) to sets of tuples (the relations) and to sets of relations simultaneously.

Double-union, double-intersection and double-minus enjoy an important property: for each relation $R$ of the result, there is at least one relation in one of the input sets whose schema is a superschema of that of $R$. This property will prove useful when application queries must be reduced to context-aware views. Indeed:

**Theorem 1.** Let $A$ and $B$ be two sets of tailoring expressions over a set of relations $\{R_1, \ldots, R_n\}$. Then $A \cap B, A \cup B$, and $A \setminus B$ are also sets of tailoring expressions over $\{R_1, \ldots, R_n\}$.

**Proof.** It follows easily from the definitions of tailoring expressions and of the three operators, also taking into account the properties of re-join. □

We now introduce some definitions and show some properties related to the relationships between tailoring expressions.

**Definition 3.** Given two tailoring expressions $e_A^R$ and $e_B^R$ over an origin $R$ of a database schema $D$, we say that $e_A^R \subseteq e_B^R$ if $\operatorname{Att}(e_A^R) \subseteq \operatorname{Att}(e_B^R)$ and $e_A^R(R) \subseteq \Pi_{\operatorname{Att}(e_A^R)}(e_B^R(R))$, for all instances $I$ of $D$.
The operators proposed in this section are instrumental to automate the “View Composition” Phase of Fig. 1, and as such will be used in Section 6. We next proceed with the description of the three fundamental phases of the CARVE methodology, starting from the first one.

5. The context model

In the first phase of the CARVE methodology, “Application Design”, beside the usual activities of software and data design, the designer carries out the identification of the aspects characterizing the application contexts; in order for the latter activity to be performed independently of the other two, the Context Dimension Tree (CDT) model is expressly designed for representing contexts.

As widely observed in the rich literature on context-aware systems [7,36], context is normally analyzed by considering its various dimensions—i.e., the points of view, or perspectives, from which the system’s situation can be observed. Besides, almost all authors agree that the analysis can be further refined by organizing these dimensions into a hierarchy.

5.1. The Context Dimension Tree

The Context Dimension Tree is a rooted labeled tree \( T = \langle N, E, r \rangle \) that describes all the possible contexts for a target scenario (see Fig. 8 for the PYL case). The tree has a root, \( r \), and the set of nodes \( N \) is partitioned into the subsets \( N_D \) and \( N_C \), that are dimension nodes –\( N_D \), black–, and concept nodes –\( N_C \), white–, representing dimensions’ values. The root \( r \) of the tree is a concept node: it models the “most general context”, thus corresponds to the entire dataset before tailoring.

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 of Fig. 8, the top dimensions are the database user’s role, his or her interface in accessing data, the current interest-topic, and the location of interest. A dimension value can be further analyzed with respect to different viewpoints (subdimensions), generating a subtree in its turn. For example, the interest-topic orders can be analyzed with respect to the order status or the order type, and these different levels of granularity for analyzing a contextual perspective will be associated with different views of the dataset, i.e., to smaller views when a concept is more specific.

Each node of the Context Dimension Tree is characterized by its type (dimension or concept) and its label, and can be uniquely identified by means of the unique path from the root to the node itself; however, in order to simplify the notation, throughout the paper we suppose that a node be identified by its label. The tree edges are not labeled.

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: \( \forall e = \langle n, m \rangle \in E \), either \( n \in N_D \land m \in N_C \) or \( n \in N_C \land m \in N_D \); i.e., a dimension node has concept nodes as children, and a concept node has dimension nodes as children.
It is possible to add one or more parameters to concept nodes and leaf dimension nodes. We introduce two more sets: $\mathcal{N}_D$, and $\mathcal{N}_C$, such that $\mathcal{N}_D \subseteq \mathcal{N}_O$, $\mathcal{N}_C \subseteq \mathcal{N}_C$, which identify, respectively, those dimension and concept nodes that have parameters. Moreover, given a node $n$, we denote by set_of_par($n$) the set of parameters associated with $n$, and by arity($n$) the cardinality of this set. A concept node parameter indicates how to select a specific set of data instances. As an example, consider the parameter date range associated with the orders interest topic: it is used, at run-time, to select exactly the date related to orders in a specific time interval. As another example, the two parameters associated with the withinXmiles location value are used to specify the current position of the user (pid) and the width of the area to be considered around such point (mid). Moreover, on concept nodes, the set of parameters of a descendant is inherited from those of its ancestors, thus $\forall n, m \in \mathcal{N}_C$ such that $m$ is in the sub-tree rooted in $n$, set_of_par($n$) $\subseteq$ set_of_par($m$).

Black nodes feature at least one white child or a parameter node, whose possible values represent as many white nodes, since a context dimension without values does not make sense. The adoption of parameters does not modify the expressive power of the model: rather, it makes it more readable, and more usable for the designer. For example, for the cost dimension we should indicate an potentially infinite number of values, thus, the presence of a parameter is more appropriate.

Dimension nodes without concept children must have exactly one parameter, i.e., $\forall n \in \mathcal{N}_D$ such that $\exists e = \langle n, m \rangle \in E$ then $n \in \mathcal{N}_D$ and arity($n$) $= 1$. Dimension nodes with concept children do not have a parameter, i.e., $\forall n \in \mathcal{N}_D$ such that $\exists e = \langle n, m \rangle \in E$ then $n \in \mathcal{N}_O$ (i.e. arity($n$) $= 0$).

The CDT has been formally presented in [10]; it can be specified in several ways, for instance we have implemented both an ontological [20] representation in OWL [37] and an XML representation based on a DTD [21]. In the following we illustrate how, given an application scenario, the possible contexts are built starting from the Context Dimension Tree and its nodes.

Given a Context Dimension Tree $T = \langle N,E,r \rangle$, a context is a conjunctive formula $\land (d_{\text{name}} = value_i)$ where each $d_{\text{name}}$ is the label of a (sub-)dimension, and each value$_i$ is a value for that dimension (i.e. the label of one of its children), assuming one of the following forms:

$\land (\text{concept\_name}(p_{\text{name}} = p_{\text{value}}))$

$\land (p_{\text{name}} = p_{\text{value}})$

A statement $d_{\text{name}} = value$ is called a context element, thus a context is a conjunction of context elements.

When a dimension is not instantiated, its default value is “ALL”.

In the View Composition Phase of CARVE each context $c$ is associated with a view over the logical schema of the entire database, corresponding to the portion of data which has to be made available to the user when, at run-time, he or she is in the context $c$. In this circumstance, any parameter value involved in the specification of that context is appropriately instantiated.

For example, the context:

$C \equiv (\text{role} = \text{customer}($\$\text{cid} = \text{"Smith"})$)

$\land (\text{interest\_topic} = \text{orders})$

$\land (\text{interface} = \text{smartphone})$

represents the situation of the customer Smith ($\$\text{cid} = \text{"Smith"}$) who is interested in receiving information about his/her orders on a smartphone. The context $C$ is composed of three context elements. During the View Composition Phase an appropriate view $v$ is assigned to $C$; at run time, when the customer Smith interacts with the PYL database via smartphone, and is interested in orders, the system will grant, to him or her, access to $v$.

Note that the value of a (sub-)dimension can be either the label of a concept node (e.g. clients), or a concept node with an additional filtering expression specified by the value of its parameters (e.g., withinXmiles($$\$\text{pid} = \text{P$\text{id}$\_\text{VALUE}}, $$\$\text{mid} = \text{M$\text{id}$\_\text{VALUE}}$)) or customer($$\$\text{cid} = \text{C$\text{id}$\_\text{VALUE}}$), or the value for a parameter of a sub-dimension (black) leaf node (e.g. $$\$\text{costid} = \text{C$\text{ost}$\_\text{VALUE}}$, where $\$\text{costid}$ is the parameter of the cost dimension). Parameter instances are fed to the system at run-time. For example, in context $C$ above the instantiation of the specific customer is based on the value “Smith".
appropriately fed to the system at run time by the application running on the smartphone.

When the designer does not want to filter the data using an explicit (and known in advance) value of a parameter \( p\_name \) related to – or inherited from – a concept node, he or she can set \( p\_name = \text{ALL} \). As a convention, when the parameter is not specified its default value is “\text{ALL}”.

Note that the possibility to specify a concept node with more than one parameter is introduced in this work to allow the designer to filter the concept instances with respect to more than one perspective, which can also assume the “\text{ALL}” value. The same effect can be obtained also in another way, by adding to the concept node some (leaf) sub-dimensions, each corresponding to a parameter, however, there are some differences in their recommended usage. The parameter of a concept node has a mandatory character (possibly set to the “\text{ALL}” value) and is inherited by all contexts which include this concept or one of its descendants.

The use of sub-dimensions allows the designer to further develop the tree and the hierarchy, an operation not allowed when using parameters. Moreover, from a practical point of view, the introduction of various leaf sub-dimensions, instead of parameters, leads to the definition of different views and consequently a higher number of contexts. On the contrary, the introduction of parameters allows the designer to define views that contain parameters themselves, which will be instantiated at run-time, allowing more dynamicity.

A general constraint on context formulation in any Context Dimension Tree is that a context must not contain two context elements \((d\_name_i = \text{value}_i)\) and \((d\_name_j = \text{value}_j)\) such that

- \(\text{value}_i\) and \(\text{value}_j\) are labels of sibling white nodes from the same parent dimension (and thus, \(d\_name_i = d\_name_j\)) or that descend from a common ancestor dimension but not from a common ancestor value other than the root.
- \(\text{value}_i\) and \(\text{value}_j\) are labels of white nodes that are one a descendant of the other (and thus, \(d\_name_i\) is descendant of \(d\_name_j\), or vice versa).

Except for this constraints, the model provides the possibility of composing contexts by selecting nodes at different levels in the tree, yielding different levels of granularity in context composition, and thus also in the tailoring of the entire data set. On the other hand, we shall see below that the designer may specify constraints on context construction that are derived from the application problem semantics, when some dimensions are not fully orthogonal.

Another flexibility aspect is 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 context does not filter the data for these dimensions, and consequently, contains the data relevant for all the values of the non-instantiated dimensions. Intuitively, contexts that are not completely specified are more general (their view is “wider”) than contexts where each dimension assumes a value.

5.2. Orthogonality and application constraints in the Context Dimension Tree

It might well happen that two CDT dimensions be non-orthogonal, meaning that the views associated with one dimension’s values might be influenced by the values assumed by the other one. For example, in the PYL application any operational context where someone with the taxi\_co role is interested in the food information does not make sense.

These are essentially two sources for non-orthogonality: on one hand, as in the previous example, given two context elements, any combination that contains them does not yield a context that is meaningful in the application domain; on the other hand, it may be that the presence of the children of a certain dimension in a context somehow “influences” the contents of the views attached to the children of another one. An example might be an application similar to PYL where the food interest topic be dependent on the category of customers.

To solve the first problem, in this section we enrich our tree model with the possibility to express application constraints over the CDT.

To solve the second problem, in Section 6.3 we allow the designer who feels that he or she is unable to design some dimensions of the CDT in an orthogonal way to define cumulative partial views, that is, partial views for more than one context element, which should replace the partial views associated with all the children of the non-orthogonal dimensions in those contexts where they appear together.

Once the Context Dimension Tree has been designed, in principle all possible contexts could be combinatorially generated as conjunctions of context elements. However, as remarked above, not necessarily all these automatically generated contexts make sense for the given application scenario, for example in PYL we would like to prevent context

\[
C \equiv (\text{role} = \text{guest}) \land (\text{interest_topic} = \text{orders})
\]

since an unregistered user cannot place orders.

The general form of a constraint is

\[
\neg(\bigwedge \text{context_element_proposition})
\]  

(1)

where a context_element_proposition is either a context element or a disjunction thereof. In constraint formulas, \textit{variables related to parameters are universally quantified}. This simple language allows the expression of many useful constraints; for instance when, in the presence of some context elements, we want to prevent the introduction of another one, we can use \textit{IF ce1 AND ce2 THEN \neg ce3} which corresponds to the formula \(\neg(\text{ce}_1 \land \text{ce}_2 \land \text{ce}_3)\).

A designer can introduce constraints for different reasons. In the following we show some categories of constraints that can be written by this simple language and that, during our experiments with the CARVE methodology and tool, proved to be the most common. To improve the readability of these common constraints we introduce a few shorthands.
Notation. Given a context element $ce = (d_{name}_i = value_i)$, we introduce:

1. a shorthand for the descendants of $ce$: $\text{Desc}(ce) \equiv \bigvee_{j} (d_{name}_j = value_j)$ (with $d_{name}_j$ (sub-)dimension in the subtree rooted in $d_{name}_i$);
2. a shorthand for $\text{Desc}(ce)$ with the addition of $ce$ itself: $\text{Desc}(ce) = \text{Desc}(ce) \vee ce$.

Throughout the paper, with abuse of notation, we will write $ce \in \text{Desc}(ce)$ to indicate that a context element $ce$ is a disjunct of $\text{Desc}(ce)$ and to simplify the discussion we will refer to them as descendants.

Here are some commonly used constraints:

- **useless-context** constraints: they allow the context designer to specify context element combinations that are not significant, i.e. that represent semantically meaningless context situations or are irrelevant for the application. In the example above, the PYL application does not allow the taxi_co role to be associated with the food information; this is formally specified as $\neg((\text{role} = \text{taxi_co})(\text{Bid} = "\text{ALL}"))$.

The $\text{Desc}$ shorthand specifies that the designer wants to discard all the contexts containing the taxi_co role combined with the interest topic food or with any of its descendants.

In general, a useless-context constraint involves $n$ context elements $(d_1 = v_1), \ldots, (d_n = v_n)$, where $v_i$ are values of the (sub-)dimensions $d_i$, and has the following form:

$\neg(\text{Desc}(d_1 = v_1) \land \cdots \land \text{Desc}(d_n = v_n))$

by means of which all the contexts where the context elements in the subtree rooted in $d_i$ occur together with the context elements derived from the subtree rooted in $d_j$ are ruled out. The notation used by the designer to write a useless-context constraint is $\text{uc}(d_1 = v_1), \ldots, (d_n = v_n)$, thus the above sample constraint is written as

$\neg((\text{role} = \text{taxi_co})(\text{Bid} = "\text{ALL}")), (\text{interest_topic} = \text{food})$

- **dimension-independent** constraints: consider a context containing a specific restaurant($\text{Bid}$) role and a $ce$ for the location dimension. Since views associated with the specific restaurant will only contain data strictly related to that restaurant (e.g., its customers, meals, etc.), whatever location $ce$, combined with it, will be associated with the same view. In general when, like in this case, the values of a (sub-)dimension do not influence the views associated with contexts containing some values for other dimensions, and therefore it is not advisable to generate different contexts all associated with the same portion of data, we can use a dimension-independent constraint. The dimension-independent constraint for the case above is represented as

$\text{dind}((\text{role} = \text{restaurant})(\text{Bid} = \text{RESTID}),$ $\text{location}) \equiv \neg((\text{role} = \text{restaurant}\text{Bid} = \text{RESTID}),$ $\text{location}) \neg((\text{role} = \text{restaurant}$

In general, a dimension-independent constraint is specified between a context element $(d_i = v_i)$ (or a parameter of a leaf dimension node) and a dimension $d_j$ (with $i \neq j$) and is formalized as follows:

$\text{dind}(d_i = v_i, d_j) \equiv
\neg((\text{Desc}(d_i = v_i) \land \bigvee_{j} (\text{Desc}(d_j = v_j)))$

When the designer specifies a dimension-independent constraint between a context element $(d_i = v_i)$ and a dimension $d_n$, then it means that all the final views of the contexts where $(d_i = v_i)$ appears together with one of the value of the $d_n$ dimension, are equivalent to one another. As a consequence, all allowed contexts containing the context element $(d_i = c_i)$ should not contain any context element for the dimension $d_n$ or any of its descendants.

- **preferred-detail** constraints: they allow the designer to express the level of detail to be preferred for a dimension when in conjunction with another, given one. An example is the one at the beginning of this section, where we assume that the restaurant role be interested in the whole food interest topic, thus a context containing the restaurant node together with a descendant of food would be too restrictive.

$\text{pd}((\text{role} = \text{restaurant}($Bid = \text{RID}),$ $\text{Desc}(\text{interest_topic} = \text{food}) =
\neg((\text{role} = \text{restaurant}($Bid = \text{RID}) \land$ $\text{Desc}(\text{interest_topic} = \text{food}))$

In general, a preferred-detail constraint is specified between context elements related to two concepts (or parameters of leaf dimension nodes) $v_i$ and $v_j$, where $v_i$ and $v_j$ are children of the (sub-)dimensions $d_i$ and $d_j$ (with $i \neq j$), respectively, and has the form:

$\neg((d_i = v_i) \land \text{Desc}(d_j = v_j))$

This kind of constraint allows the designer to discard all the contexts where $(d_i = v_i)$ occurs together with the context elements in the subtree rooted in $v_j$, excluding $v_j$. Indeed, the constraint expresses a preference on the contexts containing $(d_i = v_i)$ and a value for the dimension $d_j$ together.

Beside these three main kinds of constraint, we might need to add other, generic ones, like for example, in our application scenario, we could choose not to admit unregistered customers i.e. we do not allow the “ALL” value for the parameter of customer. This further constraint is written as $(\neg(\text{role} = \text{customer}($Bid = ALL)))$.

Another possible constraint in our PYL application imposes that the interest_topic dimension must always assume a value: $\neg(\text{interest_topic} = \text{ALL})$.

The introduction of the main classes, and of constraints in general, allows us to reduce significantly the number of
contexts (and as a consequence the number of views) to be manually, or semi-automatically, generated.

6. View definition and composition

This section presents the core of CARVE, that is, the approach for deriving a view over the global database for each significant context, specified by means of a Context Dimension Tree. In the past, two different approaches had been introduced [8]: a context-based one (also called configuration-based in [21]) and a node-based one. The former consists in manually specifying, for each allowed context, the view on the global database; as a consequence, data are tailored with a high precision, yet the approach is quite expensive in terms of the designer’s effort, since the number of allowed contexts is usually high, as discussed in Section 5. The latter approach, whose dynamics are anticipated in Fig. 1, consists of two main steps:

- Partial view definition: a partial view, expressed as a set of tailoring expressions, is associated with each context element (Phase 2 in Fig. 1); this association process can be made more or less automatic, as discussed later in the paper.
- View composition: given a context C, the partial views associated with the involved context elements are combined, by means of double-intersection, producing the final, context-aware view specifying the data relevant for C (Phase 3 in Fig. 1).

The designer may eventually revise the automatically generated views and add, remove or aggregate specific information, should the automatic procedure have inappropriately discarded or added information, or represented it with too fine a grain.

We recall that the (final) view associated with a context can be either materialized on the user’s device or maintained as a set of virtual relations; this aspect will be discussed later on in the paper.

6.1. Partial view definition

Let \( C \in \mathcal{E} \) be the set of all context elements of a given Context Dimension Tree.

In this step, the designer annotates the Context Dimension Tree associating each context element with the partial view assigned to it via the mapping

\[
\mathcal{R}_{el} : C \in \mathcal{E} \rightarrow \mathcal{V}(\mathcal{V})
\]

where \( \mathcal{V} \) is the set of all possible tailoring expressions over the global database schema we are going to tailor, as defined in Section 4, and \( \mathcal{V}(\mathcal{V}) \) is the power set of \( \mathcal{V} \). Thus, given a context element \( ce \), the partial view associated with \( ce \) is a set of tailoring expressions.

When defining partial views, the designer should observe the following restrictions:

- (a) for each expression \( e^R \) of the form \( II_X \) exp, the subset of attributes \( X \) contains the primary and foreign key attributes of \( \text{Att}(R) \);
- (b) for each \( R \) of the global schema, a partial view may contain at most one \( e^R \) (i.e. at most one expression with origin \( R \)).

These two requirements, together with the definition of tailoring expression, which restricts the possible operations to a proper subset of relational algebra (e.g. the join operator is not included), are imposed at design time because partial views are meant to contain “filtered versions” of the relations of the global database. Thus, requirement (a) is introduced because keys are necessary to identify and possibly connect tuples of two expressions. Requirement (b) implies that, if the designer wishes to include into a partial view two expressions \( e^R_1 \) and \( e^R_2 \), then they should be combined by re-join into a unique relation, having as schema the union of the two schemata. For example, if the designer wants to include into \( V \) the two expressions \( e^R_1 = \pi_X(A \bowtie B) \) and \( e^R_2 = \pi_Y(A \bowtie C) \), then the two expressions must be combined and the partial view should contain the expression \( e^R_1[\mathcal{R}e^R_2] = \pi_{X,Y}(A \bowtie B) \cup (A \bowtie C) \).

In this perspective, the partial view associated with a context element has a tailored schema with respect to the global one, where some relations do not appear and those that appear may have a reduced number of attributes (and tuples). The same holds also for the final view associated with a context. Relation names, as well as attribute names, are left the same as the original ones. Indeed, requirement (b) of partial view definition guarantees that, by applying the double-intersection operator between two partial views, we obtain in the result at most one expression with origin \( R \), for each relation \( R \) of the global database.

This property and this naming convention guarantee that the query resilience requirement introduced in Section 2 is satisfied. As a consequence, a context-aware application need only refer to the global schema, without any need to be modified with context changes. Observe that this is much more than we normally obtain when building user or application views over a global database in the traditional database design methodology. By enforcing these constraints on view definition, views are not only dedicated to specific users and applications, but also to different contexts of usage, allowing the system to seamlessly change the view at each change of context, without affecting the application code.

As seen above, the final view for a given context is obtained by double-intersection, whose output is in general smaller than the input sets. Thus, the partial view associated with a context element \( ce \) should be designed in such a way as to contain all the portions of relations the designer consider relevant for \( ce \), including additional relations that are connected one to the other by means of foreign key constraints and are useful to complete the selected information. As a conclusion, the view for \( ce \) should only exclude information that is not related to the context element \( ce \).

\(^2\) Note that the two schemata have a non-empty intersection because at least \( R \)’s primary key must be present in both.
For example the following partial view for the orders

\( \text{Rel}((\text{interest\_topic} = \text{orders})) = \)

\{\text{ORDERS, ORDERDETAILS, PICKUP,}

\text{CUSTOMERS} \leftarrow \text{ORDERS, RESTAURANTS,}

\text{RESTAURANTCUISINE, CUISINE}\}

includes all the information that is related, in some way, to orders; that is, besides the ORDERDETAILS, which contains all and only information on the composition of the orders, the view also contains other useful information about customers who made some orders, restaurants and their details and pick-up locations.

Consider now the following partial view, associated with the customer role (the specific customer is identified by the parameter value), which includes all the possible information useful for that customer in the PYL scenario:

\[
\text{Rel}(\text{role} = \text{customer}(\text{Scid} = \text{CID})) = \{\text{CUISINES, DAILYSPECIALS, DELIVERY} \leftarrow \text{\sigma}_{\text{CUISINE}} = \text{Scid ORDERs, DELIVERYCOST, DISHES, DISHCATEGORY, DISHINGREDIENT, DRINKS, INGREDIENTS, MENUS, MENUDElIES, SCID} \leftarrow \text{\sigma}_{\text{CUISINE}} = \text{Scid ORDERs, ORDERDETAIL, RE:R0TASIONS, RESTAURANTCUISINE, RESTAURANTDRINK, RESTAURANTSERVICE, SERVICES, USERS} \leftarrow \text{\sigma}_{\text{CUISINE}} = \text{Scid CUSTOM:RS, ZONES, PICKUP}\}
\]

The view for the context

\[ C \equiv (\text{role} = \text{customer}(\text{Scid} = \text{CID})) \wedge (\text{interest\_topic} = \text{orders}) \]

is obtained as

\[ \text{Rel}(C) = \text{Rel}((\text{role} = \text{customer}(\text{Scid} = \text{CID}))) \wedge \text{Rel}((\text{interest\_topic} = \text{orders})) = \{\text{RESTAURANTS, RESTAURANTCUISINE, CUISINE, PICKUP, \sigma}_{\text{CUISINE}} = \text{Scid ORDERs, ORDERDETAIL, \sigma}_{\text{CUISINE}} = \text{Scid ORDERs}\} \]

We now make some considerations related to the relationships between partial views associated with elements of the same Context Dimension Tree.

We say that a context element \( ce_i \) is more specific than \( ce_j \) (or, equivalently, that \( ce_j \) is more abstract than \( ce_i \)), and write \( ce_i < ce_j \) (or \( ce_j > ce_i \)), if and only if \( ce_i \) is a disjunct of \( \text{Desc}(ce_j) \), and introduce the following assumption, used throughout this paper.

**Assumption 1.** For each pair of context elements \( ce_1 \) and \( ce_2 \) in a Context Dimension Tree, if \( ce_1 > ce_2 \) then \( \text{Rel}(ce_1) \supseteq \text{Rel}(ce_2) \).

The rationale behind **Assumption 1** is the following: the hierarchical structure of the Context Dimension Tree suggests that the detail level adopted to select data grows while descending the sub-tree related to each context dimension. Thus, we deem it desirable that the partial view for a context element of a white node \( n \) contain the partial views of the context elements related to the descendants of \( n \).

To satisfy **Assumption 1**, CARVE imposes that, given two context elements \( ce_1 \) and \( ce_2 \) in a Context Dimension Tree such that \( ce_1 > ce_2 \), and assuming the designer has specified \( \text{Rel}(ce_1) = \{R_1, \ldots, R_k\} \), then the relevant area \( \text{Rel}(ce_2) = \{S_1, \ldots, S_k\} \), with \( \leq k \), is such that for each relation \( S_i \) in \( \text{Rel}(ce_2) \), there exists a \( R_j \) in \( \text{Rel}(ce_1) \), such that \( S_i \subseteq R_j \).

**Example 4.** Let us motivate **Assumption 1** and our methodological approach by means of an example. It is quite reasonable to suppose that the partial view for the vegetarian cuisine further restricts \( \text{Rel}((\text{interest\_topic} = \text{food})) \) only to information about vegetarian dishes (isVeget = T). Thus, if we consider as partial view for the food interest-topic the following set of tailoring expressions:

\[
\text{Rel}((\text{interest\_topic} = \text{food})) = \{\text{CUISINES, DAILYSPECIALS, DISHES, DISHCATEGORY, DISHINGREDIENT, DRINKS, INGREDIENTS, MENUS, MENUDElIES, RESTAURANTS, RESTAURANTCUISINE, RESTAURANTDRINK, PICKUP, ZONES}\}
\]

The partial view for vegetarian cuisine, satisfying **Assumption 1**, can be reasonably assigned as follows, by further reducing the tailoring expressions of \( \text{Rel}((\text{interest\_topic} = \text{food})) \):

\[
\text{Rel}(\text{(cuisine} = \text{vegetarian})) = \{\text{CUISINES} \leftarrow \text{RESTAURANTCUISINE} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, DAILYSPECIALS} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, DISHCATEGORY} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, DISHINGREDIENT} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, DRINKS, INGREDIENTS} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, RESTAURANTS} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, MENUS} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, MENUDElIES} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, RESTAURANTCUISINE} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, RESTAURANTDRINK} \leftarrow \text{\sigma}_{\text{isVeget} = \text{T}} \text{DISHES, PICKUP, ZONES}\}
\]

The partial order between context elements induces a partial order between contexts, as formalized in the following definition:

**Definition 5.** Let us consider two contexts \( C_1 = c_{e_11} \land \cdots \land c_{e_1n} \) and \( C_2 = c_{e_21} \land \cdots \land c_{e_2m} \), \( C_1 \) is more abstract than \( C_2 \), written as \( C_1 > C_2 \), if and only if for each context element \( c_{e_1j} \) in \( C_1 \) there is a context element \( c_{e_2j} \) in \( C_2 \) such that (i) \( c_{e_1j} > c_{e_2j} \) or \( c_{e_2j} = c_{e_1j} \), and (ii) if \( c_{e_2j} \) contains the instantiation of a parameter \( p \), then \( p \) is either undefined (ALL) in \( c_{e_1j} \) or the type of \( p \) is a set and the instantiation of \( p \) in \( c_{e_1j} \) contains (or is equal to) the instantiation of \( p \) in \( c_{e_2j} \).

For example, the context \( C_1 \equiv (\text{role} = \text{customer}(\text{Scid} = \text{"Smith"})) \land (\text{interest\_topic} = \text{orders}) \) is more abstract than \( C_2 \equiv (\text{role} = \text{customer}(\text{Scid} = \text{"Smith"})) \land (\text{type} = \text{delivery}) \land (\text{interface} = \text{smartphone}) \), thus \( C_1 > C_2 \).
6.2. View-composition example

In this subsection we show a full example of the generation of a context-aware view by double-intersection of partial views.

**Example 5.** Consider the context of a customer $\text{Scid}$ interested in visualizing on her/his smartphone information about the orders s/he made to the PYL services. The current context of the customer is

$$C = \langle \text{role} = \text{customer}(\text{Scid} = \text{CID}) \rangle$$

$$\land \langle \text{interest_topic} = \text{orders} \rangle$$

$$\land \langle \text{interface} = \text{smartphone} \rangle$$

The partial views associated with the customer role and the orders interest-topic have been defined in the previous subsection. The partial view associated with the smartphone interface is the global database deprived of long descriptions and high resolution images. More precisely, we do not include in the selected schema the LONGDESCRIPTION attribute in DISHES relation, the PHOTOD1 cisely, we do not include in the selected schema the long descriptions and high resolution images. More precisely, we do not include in the selected schema the LONGDESCRIPTION attribute in DISHES, and the attribute PHOTOR1 of the RESTAURANTS relations. Let $P_1$ and $P_2$ be the sets of attributes

$$P_1 = \{\text{DishID, RestID, ShortDescription, Price, isVeget, isSpicy,}
\text{ isMildSpicy, wasFrozen, isAvailable, PhotoD2, CategoryID}\}$$

$$P_2 = \{\text{RestID, Name, RNumber, Address, ZIP,}
\text{ City, State, ZoneID, Phone,}
\text{ Fax, Email, Web, OpeningHrs, ClosingHrs,}
\text{ ClosingDay, Capacity, Parking,}
\text{ MinOrder, Rating, PhotoR2, UserID}\}$$

Then

$$\text{Rel}(\text{interface} = \text{smartphone})$$

$$= \{\text{Cuisines, Customers, DailySpecials,}
\text{ Delivery, DeliveryCost, PI, Dishes, DishCategory,}
\text{ DishIngredient, Drinks, Menus, Ingredients, MenuDishes,}
\text{ OrderDetail, Orders, Reservations, PI, Restaurants,}
\text{ RestaurantCuisine, RestaurantDrink,}
\text{ RestaurantService, Services,}
\text{ Pickup, Taxi, Zones, Users}\}$$

Thus, the final view, obtained by applying double-intersection to the partial views of all the context elements of $C$, is the following:

$$\text{Rel}(C) = \text{Rel}(\text{role} = \text{customer}(\text{Scid} = \text{CID}))$$

$$\land \text{Rel}(\text{interest_topic} = \text{orders})$$

$$\land \text{Rel}(\text{interface} = \text{smartphone})$$

$$= \{PI, \text{ Restaurants, RestaurantCuisine, Cuisine, Pickup,}
\text{ OrderDetail}, \text{ Orders, Reservations, PI, Restaurants,}
\text{ RestaurantCuisine, RestaurantDrink,}
\text{ RestaurantService, Services,}
\text{ Pickup, Taxi, Zones, Users}\}$$

The final view for the context $C$ contains all the information about orders made by the run-time selected customers, and no high-resolution images (due to the available interface) about restaurants.

6.3. Non-orthogonal dimensions in the view definition process

When the designer is not satisfied by the per-element partial view definition, because the Context Dimension Tree contains some non-orthogonal dimensions (see Section 5.2), s/he can decide to manually specify the contextual view associated with the set of context elements corresponding to the non-orthogonal dimensions. In this case we have to extend function Rel to a partial function

$$\text{Rel}^F : \varphi(CE) \rightarrow \varphi(V)$$

associating partial views with some subsets of $CE$, where $\text{Rel}^F(CE)$ is defined as equal to $\text{Rel}(CE)$. Given a set of context elements $CE$ for which $\text{Rel}^F(CE)$ has been defined by the designer, $\text{Rel}^F(CE)$ must be used in the partial view composition for any context containing all the context elements $ce_i$ of $CE$, thus replacing all the partial views $\text{Rel}(ce_i), \ldots, \text{Rel}(ce_n)$ in the double intersection $\cap_{(ce_i \in CE)} \text{Rel}(ce_i)$ with $\text{Rel}^F(CE)$.

Note that, in order to satisfy Assumption 1 specified above, when the designer manually specifies a view $\text{Rel}^F(CE)$, s/he has to define it in such a way that $\text{Rel}^F(CE) \supseteq \text{Rel}(ce_i)$, for each $ce_i \in CE$, i.e. the ad hoc view has to contain those associated with the context elements in $CE$.

6.4. Low-impact partial-view definition process

As noted in Section 6.1, and given the semantics of double-intersection, the partial-view definition phase is intended to assign the widest possible view to each context element of the Context Dimension Tree; consequently, it is natural for the designer, once this policy is selected, to perform this phase by navigating the Context Dimension Tree top-down (from the root to the leaves). This actually means that the view for the context element related to the root is the entire database, and the partial view for a context element $ce$ 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 context element $ce$ a view (Rel(ce)) over the (previously defined) view (Rel(ce')) of its parent $ce'$, according to Assumption 1.

However, to reduce the designer workload, now we introduce a semi-automatic process that acts in a bottom-up fashion: 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 system recursively composes the partial view associated with a context element $ce$ by applying double-union to those associated with its children; additional portions of the global database, deemed useful for $ce$ by the designer, may be manually added.

As a suggestion for the designer, the partial view associated with a leaf $ce$ (i.e., with one of the most specific context elements) should contain at least the portions of the global schema that are strictly related to $ce$. For example, by referring to our running example, $\text{Rel}(\text{interest_topic} = \text{orders})$ can be obtained as double-union among
\( \text{Rel}(\text{type} = \text{delivery}), \text{Rel}(\text{type} = \text{other}), \) and \( \text{Rel}(\text{status} = \{\text{open}/\text{close} = \text{OC}\}) \).

Of course, the designer may also include other relations that directly refer, through foreign key constraints, to the main relations selected.

**Example 6.** Let us now compute the partial view for the `orders` interest-topic by applying double-union to the partial views of its descendants, shown in the table below.

<table>
<thead>
<tr>
<th>Context element</th>
<th>Partial view</th>
</tr>
</thead>
<tbody>
<tr>
<td>(type = delivery)</td>
<td>( {\text{Orders} \leftarrow \text{Delivery}, \text{OrderDetails} \leftarrow \text{Delivery}, \text{Customers} \leftarrow \text{Orders} \leftarrow \text{Delivery}, \text{Restaurant} \leftarrow \text{OrderDetails} \leftarrow \text{Delivery}} )</td>
</tr>
<tr>
<td>(type = other)</td>
<td>( {\text{Delivery} \leftarrow \text{Pickup} \leftarrow \text{NULL} \text{Orders}, \text{OrderDetails} \leftarrow \text{Pickup} \leftarrow \text{NULL} \text{Orders}, \text{Customers} \leftarrow \text{Pickup} \leftarrow \text{NULL} \text{Orders}, \text{Restaurant} \leftarrow \text{OrderDetails} \leftarrow \text{Pickup} \leftarrow \text{NULL} \text{Orders}} )</td>
</tr>
<tr>
<td>(status = {\text{open}/\text{close} = \text{SO}})</td>
<td>( {\text{Orders} \leftarrow \text{Delivery}, \text{OrderDetails} \leftarrow \text{Delivery}, \text{Customers} \leftarrow \text{Orders} \leftarrow \text{Delivery}, \text{Restaurant} \leftarrow \text{OrderDetails} \leftarrow \text{Delivery}} )</td>
</tr>
</tbody>
</table>

In the partial view definition shown in the table, we have assumed that when a customer chooses the pick-up service for an order, and not the delivery at home, then the attribute `PickupID` of the `Orders` relation be NOT NULL. The resulting partial view, associated with the `orders` interest topic, is as follows.

\[
\text{Rel}(\text{interest_topic} = \text{orders})) = \text{Rel}(\text{type} = \text{delivery})) \Rightarrow \text{Rel}(\text{type} = \text{other})) \Rightarrow \text{Rel}(\text{status} = \{\text{open}/\text{close} = \text{SO}\})) = \{\text{Delivery}, \text{Pickup} \leftarrow \text{NULL} \text{Orders}, \text{Orders}, \text{OrderDetails}, \text{Customers}, \text{Restaurant} \leftarrow \text{OrderDetails} \leftarrow \text{Orders}\}
\]

For example, the view associated with the context \( C = \{\text{role} = \text{taxi_co}(\text{Scid} = \text{CID})\} \)

\( \wedge (\text{location} = \text{zone}(\$\text{zid} = \text{"CentralSt."})) \wedge (\text{interest_topic} = \text{orders}) \)

is obtained by means of the following expression:

\[
\text{Rel}(\text{C}) = \text{Rel}(\{\text{role} = \text{taxi_co}(\text{Scid} = \text{CID})\} \\
\wedge \text{Rel}(\text{location} = \text{zone}(\$\text{zid} = \text{"CentralSt."})) \\
\wedge \text{Rel}(\text{interest_topic} = \text{orders})
\]

Within CARVE, a few properties useful for the final view-composition phase can be derived.

### 6.5. View-composition properties

The compositional operators used to obtain a context-dependent view enjoy some important properties. The next two theorems show how, given the view for a context \( C = \bigcup_{i=1}^{k} (d\_name_i = \text{value}_i) \), we can obtain the view for a context \( C = \bigcup_{i=1}^{k} (d\_name_i = \text{value}_i) \) (containing an additional context element \( c_{k+1} = (d\_name_{k+1} = \text{value}_{k+1}) \)) by simply composing, by double-intersection, the partial view of \( c_{k+1} \) with \( \text{Rel}(C) \).

**Property.** Let \( \text{Rel}(C_1) \) be the view defined for the context

\[
C_1 = \bigcup_{i=1}^{k} (d\_name_i = \text{value}_i)
\]

and \( C_2 = \bigcup_{i=1}^{k+1} (d\_name_i = \text{value}_i) \) another context, which differs from \( C_1 \) only for the last context element. Then

\[
\text{Rel}(C_2) = \text{Rel}(C_1) \delta \text{Rel}(d\_name_{k+1} = \text{value}_{k+1})
\]

The property follows from the commutativity and associativity of \( \delta \), by applying the definition to a set of \( k+1 \) operands.

**Theorem 3.** Let \( \text{Rel}(ce) \) be the partial view for a context element \( ce \), defined by composing, with double-union, the partial views \( \text{Rel}(ce_1), \ldots, \text{Rel}(ce_n) \) of its children \( ce_1, \ldots, ce_n \). If we add another child \( ce_{n+1} \) to \( ce \), then the partial view becomes \( \text{Rel}(C) \psi \text{Rel}(ce_{n+1}) \).

**Proof.** From the commutativity and associativity of \( \psi \). \( \square \)

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

**Theorem 4.** Let \( C = \bigcup_{i=1}^{k} (c_{e_j} \wedge c_{e_j} \wedge \ldots \wedge c_{e_j} \wedge c_{e_j} \wedge \ldots \) be two context elements such that \( c_{e_j} \wedge c_{e_j} \wedge k \leq j \leq k \). Then \( \text{Rel}(C) = \text{Rel}(c_{e_1} \wedge \ldots \wedge c_{e_j} \wedge \ldots \wedge c_{e_j} \wedge \ldots \wedge c_{e_j}) \) that is, by replacing a context element with a more general one we obtain a view which contains that of the initial context.

**Proof.** Let us recall that the composition of partial views is performed by means of the double-intersection operator. According to Assumption 1, since \( c_{e_j} < c_{e_j} \), then \( \text{Rel}(c_{e_j}) \subseteq \text{Rel}(c_{e_j}) \). Hence,

\[
\text{Rel}(C) = \text{Rel}(c_{e_1}) \cap \ldots \cap \text{Rel}(c_{e_j}) \cap \ldots \cap \text{Rel}(c_{e_j})
\]

\( \Rightarrow \text{Rel}(c_{e_1}) \cap \ldots \cap \text{Rel}(c_{e_j}) \cap \ldots \cap \text{Rel}(c_{e_j}) \)

\[
\Rightarrow \text{Rel}(c_{e_1} \wedge \ldots \wedge c_{e_j} \wedge \ldots \wedge c_{e_j}) \square
\]

**Theorem 5.** Let \( c_{e_j} \) and \( c_{e_j} \) be two context elements such that \( c_{e_j} < c_{e_j} \) and assume we compute the partial views of internal nodes by using the double-union operator (\( \psi \)). Then \( \text{Rel}(c_{e_j}) \psi \text{Rel}(c_{e_j}) \).

**Proof.** Follows immediately from the monotonicity of double-union w.r.t. \( \subseteq \). \( \square \)

In the next subsection we discuss how constraints, in particular **useless-context** ones, affect partial view composition, and we show the use of the double-minus operator to support the compositional process.
6.6. Effect of the useless-context constraints in the view definition process

As described in Section 5, the Context Dimension Tree can be enriched with constraints for discarding semantically meaningless configurations. In particular, useless-context constraints are introduced to avoid contexts that make no sense or have to be prohibited for the target application. For this reason, care has to be taken when defining the view for a context \( C \) containing context elements for which a useless-context constraint has been introduced. In this section we analyze how the presence of useless-context constraints between nodes included in a context \( C \) can affect the composition of partial and final views. The double-minus operator perfectly serves the purpose to define the views in the presence of useless-context constraint.

Let us introduce a few notations.

**Notation.** Given a context element \( ce \equiv (d_{name}, value) \),

(a) the set of “siblings” of \( ce \) in the Context Dimension Tree is

\[
\text{Sib}(ce) = \{ (d_{name}, value) \mid \exists \langle value, d_{name} \rangle \in E \text{ with } value \neq value_i \}
\]

(b) the “father” of \( ce \) is \( \text{father}(ce) = ce' \) with \( ce' = (d_{name}, value) \) being \( d_{name} \) the unique dimension node s.t.

\[
\exists \langle value, d_{name} \rangle \in E
\]

Consider a useless-context constraint \( uc(ce_1, \ldots, ce_n) \). Notice that, in fact, this constraint means that for each \( i \) the partial view of \( ce_i \) should never be combined with the partial views of \( ce_1, \ldots, ce_{i-1}, ce_{i+1}, \ldots, ce_n \).

We will see that the partial view associated with a context element which is mentioned in a useless context constraint may influence the definition of the partial views associated with its ancestors and to its siblings.

To simplify our discussion, let us now consider the effects on the partial view for a context element \( (d_2 = a) \) when one of its children \( (d_3 = f) \) is mentioned in a useless-context constraint \( uc(ce_1, \ldots, (d_3 = f), \ldots, ce_n) \) (e.g. see the portion of Context Dimension Tree on the left of Fig. 9).

We remind the reader that, by Assumption 1, the partial view of a context element contains those of its descendants, thus, without loss of generality, one of the three cases depicted in Fig. 9 is possible. Fig. 9 shows, the situation for the sibling \( e \) of \( f \). \( Rel(d_2 = a) \) contains \( Rel(d_3 = e) \) as well as \( Rel(d_3 = f) \), and the three cases show the possible containment situations between \( Rel(d_3 = e) \) and \( Rel(d_3 = f) \), that the designer may have conceived (in cases A and B one area is fully contained in the other one, in case C the two areas are incomparable).

While working bottom-up by means of the low-impact partial view definition procedure of Section 6.4, the definitions of \( Rel(d_2 = a) \) and \( Rel(d_3 = s_i) \), for each \( (d_3 = s_i) \in \text{Sib}(d_3 = f) \) should be changed to take into account the fact that, if \( (d_3 = f) \) has been excluded from some context by a useless-context constraint, its view should not be included “by mistake” in the views related to contexts, where its father \( (d_2 = a) \) or any of its siblings appears. Thus, when designing the views for contexts of the form \( d_1 = k \land d_2 = a \land \cdots \) (or \( d_1 = k \land d_2 = s_i \land \cdots \)), we should use modified versions of \( Rel(d_2 = a) \) and \( Rel(d_3 = s_i) \).

Once the partial views for the context element \( (d_2 = a) \) and for each sibling \( (d_3 = s_i) \) of \( (d_3 = f) \) have been defined according to the low-impact partial view definition procedure, the designer has to modify them as depicted in Fig. 10, according to the following definitions:

\[
\overline{Rel}(d_2 = a) = \begin{cases} 
Rel(d_2 = a) & \text{if } Rel(d_3 = f) \subset Rel(d_3 = s_i) \land (d_1 = k \land d_2 = a) \\
Rel(d_2 = a) \cup (Rel(d_2 = a) \cap Rel(d_3 = s_i)) & \text{if } Rel(d_3 = f) \subset Rel(d_3 = s_i) \\
\emptyset & \text{otherwise} \end{cases}
\]

That is, if the portion of information related to the context element \( (d_3 = f) \) is strictly contained into the view related to its sibling \( (d_3 = s_i) \), we remove \( Rel((d_3 = f)) \) from \( Rel((d_2 = a)) \). In the other cases, we remove from \( Rel((d_2 = a)) \) only the portion of \( Rel((d_3 = f)) \) that is not contained into that of any other of its siblings.

Moreover, the partial view of the context element \( (d_3 = s_i) \) changes as defined in the following:

\[
\overline{Rel}(d_3 = s_i) = \begin{cases} 
Rel((d_3 = s_i)) & \text{if } Rel((d_3 = s_i)) \subset Rel((d_3 = f)) \\
Rel((d_3 = s_i)) \cap Rel((d_3 = f)) & \text{otherwise} \end{cases}
\]

**Example 7.** Consider the context

\[
C \equiv (\text{role} = \text{restaurant}(\text{rid} = \text{RID}))
\land (\text{interest_topic} = \text{food})
\]

and suppose the following partial views have been specified:

\[
\overline{Rel}((\text{interest_topic} = \text{food})) = \{ \text{Cuisines}, \text{DailySpecials}, \text{Dishes}, \text{DishCategory}, \text{DishIngredient}, \text{Drinks}, \text{Ingredients}, \text{Menus}, \text{MenuDishes}, \text{RestaurantCuisine}, \text{Restaurants}, \text{RestaurantDrink}, \text{Pickup}, \text{Zones} \}
\]

\[
\overline{Rel}((\text{information} = \text{restaurants})) = \{ \text{Restaurants}, \text{RestaurantCuisine}, \text{Cuisines}, \text{Zones} \}
\]

![Fig. 9. Partial view selection: possible cases.](image-url)
following we present the tool. The assessment we have carried out, also by means of CADDFrame, to evaluate the effectiveness of CARVE and estimate its complexity is discussed in Section 8.

7.1. CADDFrame: a prototype tool to support CARVE

CADDFrame is composed by two fundamental modules: the first one, Context Space Designer (CSD), supports the design of the Context Dimension Tree, while the second one is the newly developed Relational Tailoring Assistant (RTA). The RTA lists all the context elements of the CDT designed by means of CSD, and allow the designer to assign a partial view to each one of them. A screenshot of the tool is shown in Fig. 11, where the list of context elements for which a partial view need be defined appears in the left-hand side of the window panel (area 1); in our running example, the number of context elements is 24.

For each context element, the designer can select the interesting relations in the database, and for each one of these it is possible to select a (not necessarily proper) subset of the attributes, as well as to impose (complex) conditions on the attribute values by means of a selection or a semi-join. The partial-view construction for the internal context elements can also be performed by means of the low-impact procedure, via the \( \psi \) operator. The tool shows, for the context element currently under consideration, the selected relation schemata (central area 1); in our running example, the number of context elements is 24.

It may happen that the designer has specified a very detailed Context Dimension Tree, where the views do not differ too much from one another, but appear to be obtained as more or less refined selections on the same partial view. As an example, consider the status node in the PLY example; if the designer had introduced two children, say open and closed, rather than a parameter, two partial views should have been defined, differing only in the where status condition. As a consequence, we suggest that the designer explicitly list (sub)dimension values only when they reasonably lead to different views.

7. The CARVE framework prototype

We have developed a prototype tool, called CADDFrame,\(^3\) implementing CARVE and the associated operators. In the

\(^3\) CADDFrame is an extension of a preliminary tool, CADD [38], which worked on an ER schema and on a per-context basis.
rather than simply to different where clauses. This situation may actually occur, since the CDT is usually designed independently of the global schema, which may be the result of an integration process of independent, heterogeneous sources. In such cases, the designer may reshape the CDT, eventually adjusting the partial views per context element, and let the system recompute the views per context.

7.2. Partial-view definition

CADDFrame implements both strategies: (i) definition of the partial views for all context elements, starting from the parents and moving towards the leaves of the tree, followed by the application of double-intersection to compute the final views associated with each context; (ii) definition of the partial only for the leaf context elements, using double-union to generate the partial views of the other context elements, followed by the application of double-intersection to compute the final views associated with each context.

As already noted, the first strategy, although more burdensome, allows for a more refined specification of the partial views for non-leaf nodes, thus leading – in general – to a more precise final view. However, should the number of context elements be too relevant, the second strategy leads to a quicker result. This occurs when the tree is a deep one, with several levels of sub-dimensions. In both situations the user can later manually refine the final views to include or exclude information.

The prototype tool allows the designer to select the operator at any time, although it is advisable to decide in advance which operator will be used, before defining the partial views. The designer has access to the entire list of significant contexts and to the available operators; the resulting combination of partial views produces a set of views that can be reviewed and refined.

The application of CARVE to the example scenario proved effective both in the determination of the different data portions each user had access to, in the various contexts, and in the development of applications targeted to “sets of contexts” they will be used in. More precisely, it is possible to develop an “application-template” to be customized in terms of menus and available functionality based on the actual current context, thus moving to a refinement phase the detailed ad hoc features of the application.

This is the phase when the queries embedded in the applications can be assessed against the views associated with the contexts, in order to minimize the out-of-context exceptions.

The total number of valid contexts in the Context Dimension Tree of our running example is around 4850 (about 8950 if constraints are not used), a quite high number as it usually happens when introducing several details in the tree itself, to specify as many elements as possible that affect the data tailoring process. As already observed, though, the significant datum is the number of context elements, which amounts to a few tens (24 in the presented CDT). Furthermore, if the second strategy is adopted, the number of partial views to be specified by the designer is limited to leaf context elements, which are 22. The total number of contexts eventually affects the amount of refinement the designer shall apply to the automatically generated views. Our experience is that only a subset of the valid contexts actually needs to be revised, and even a smaller subset of them needs user intervention.

8. Methodology analysis and lessons learned

To assess the CARVE methodology, during the last few years we engaged various groups of master and PhD students in a set of experiments on different case studies.
of development of data-oriented context-aware systems, allowing us to perform an empirical evaluation. We empirically and qualitatively analyzed the effort required to exploit the proposed methodology by the designer with respect to the achieved benefits. The evaluation criteria we adopted are the following ones:

- number of user-defined views vs. number of valid contexts, to allow the understanding of the limited effort required to the designer;
- average number of relations per context vs. number of relations in the database, to provide an idea of the reduction in the amount of data the final user will deal with.

In all cases the methodology was applied to existing databases, meant to be accessed through dedicated applications; the outputs were customized views based on the specific contexts the application was meant to be used in. Therefore, CARVE could be exploited at its best, by allowing the designers to apply all three main steps of the methodology.

Besides the PYL case study used throughout this paper, the following cases were studied.

Wine is an application related to a networked enterprise for wine production, characterized by several user roles (e.g., oenologist, agronomist, commercial managers) accessing different information on the grapes, the environmental conditions (data are also collected from sensors), the several phases of the productive process, as well as the final distribution and sales. Besides the information related to these two main dimensions (i.e., role and interest-topic), time and space are used to further tailor data. Information is accessed through a context-aware web portal, as well as from portable devices during specialist personnel visits to the winery. The characteristics of the designed Context Dimension Tree and the related views are reported in Table 1.

The Mobile Student Assistant application scenario, developed for Politecnico di Milano’s students, faculty and visiting guests, to access the university-related information on classes, rooms, schedules and material. Since the university is distributed on two campuses, information on local transportation useful is also available. Main goal of the context-aware methodology is context-aware management of championships; different roles exist (i.e., team leader, market and system managers), different phases (i.e., regular season and market window) and different interest topics (e.g., player’s and team statistics, matches results). The CARVE methodology has been adopted for the development of a context-aware layer to tailor data and customize user’s data access based on the active user’s context.

FantaCalcio is the scenario related to a popular soccer-related game where independent groups of people simulate the Italian soccer championship, by creating a team with players during a market/exchange period, where players are acquired based on their value, performance in the previous year (e.g., the number of goals). Then, during the regular season, based on the real performance of the players, each team gets scores leading to a ranking. Each group of people plays a “local” championship and there are web-based systems supporting an electronic management of championships; different roles exist (i.e., team leader, market and system managers), different phases (i.e., regular season and market window) and different interest topics (e.g., player’s and team statistics, matches results). The CARVE methodology has been adopted for the development of a context-aware layer to tailor data and customize user’s data access based on the active user’s context.

In Table 1 we reported, for each working scenario, the number of context elements in the Context Dimension Tree, the number of leaves (column 2), the number of valid contexts excluding those prohibited by constraints (column 3), and the designer’s effort in identifying the relevant area definitions (listed in column 4 and equal to the number of context elements in the top-down methodological approach), or in manually performing a tailoring of the information in case the CARVE methodology would not be adopted.

Table 2 reports the number of relations in the entire database, hosted on the server, and the number of relations constituting the final view associated with a context, to provide an idea of the reduction of the complexity of the db schema, which includes only relevant information for each given context. Furthermore, the amount of tailoring offered by the approach can be found in the last column of the table, where we reported the ratio of the data associated with a context with respect to the amount of data of the entire database on the server. At run time, this is the actual benefit the user perceives, having to manage only a limited amount of (focused) information.

The development of these and other case studies allowed us to evaluate CARVE from a qualitative point of view and to derive some considerations discussed in the next subsection.

---

4 http://artdeco.ws.dei.polimi.it/.

<table>
<thead>
<tr>
<th>Name</th>
<th># Context elements</th>
<th># Leaves</th>
<th># Valid contexts</th>
<th># User-defined views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pickup your lunch</td>
<td>24</td>
<td>22</td>
<td>4851</td>
<td>24</td>
</tr>
<tr>
<td>Wine</td>
<td>41</td>
<td>36</td>
<td>1870</td>
<td>36</td>
</tr>
<tr>
<td>Mobile student assistant</td>
<td>22</td>
<td>20</td>
<td>633</td>
<td>22</td>
</tr>
<tr>
<td>Video on demand</td>
<td>9</td>
<td>9</td>
<td>44</td>
<td>9</td>
</tr>
<tr>
<td>FantaCalcio</td>
<td>14</td>
<td>13</td>
<td>275</td>
<td>14</td>
</tr>
</tbody>
</table>
In all cases we studied, the effort for designing the Context Dimension Tree and revising amounted at most to a half-day work, considering that the refinement process may be carried out after a preliminary view composition step based on the feedback gathered by analyzing the produced contexts and associated views. In fact, there are situations where the designer either initially adds several details that actually do not impact on the tailoring process or omits aspects needed to prune data.

As for the definition of the relevant areas associated with every context element, the activity is limited to their number, that is in the orders of tens. More precisely, when we introduced the first hints of CARVE [9] the designer effort we envisaged was really high, since the idea was to associate a view with each valid context of the Context Dimension Tree. Even for a CDT with a small number of context elements it may become a very demanding task—in the worst-case there are $2^n$ contexts. The work proposed in this paper has first simplified the problem to the manual definition of the $n$ partial views, further reducing it, by introducing the low-impact strategy for partial view definition that requires to work only on the CDT leaves. As a result, the students, who had acquired a good knowledge of the relational schema of the database they were working on, took an average of 15 min to define the partial view associated with each context element.

Another important, methodological consideration concerns the difficulty, on the designer’s part, of deciding which information constitutes the context and which constitutes the “object data”. Let us consider the running example: here the database contains the company data and we are using the context to reduce and focus these data for different contexts of use.

Note that the CDT has two main categories of dimensions. Some dimensions (typically the Interest Topic) have values that also appear in the database itself, and indeed are typically used to tailor the data by means of selections or projections of the database tables; actually, some automatic procedure might be devised to derive automatically derive the corresponding partial views. Other dimensions (like the Role or the Interface) do not necessarily refer to database attributes or values, thus the designer has to know, from the system requirements, which are the most appropriate data for the corresponding partial views.

The dimensions of the latter category are actually those that most clearly belong to the context, since their values do not necessarily belong to the database. Note that these values might well be stored in it, but, from a methodological point of view, do not seem to pertain to the database domain, exactly as the information about the user credentials or profile is used to grant and revoke access to portions of data for privacy reasons.

As for the dimensions like Interest Topic, even if their values are normally available in the database, the designer should decide whether their usage is actually orthogonal, and thus should be kept so as a support in producing external database views.

Further general considerations can be drawn from our experience:

- The context model is rather flexible and well supports the designer in the definition of the elements that affect data selection in the different situations; this flexibility may lead to a very high number of contexts, although not all of them would be actually used. As a result, the introduction and formalization of different types of constraints is a powerful instrument to avoid non-interesting contexts. Nevertheless, in the more general perspective of the methodology efficiency, the cardinality of the valid contexts with respect to the most used ones is still a limitation, especially when associating partial views also to sets of contextual elements. As with all design models and methodologies, the designers have to acquire, during their experience, an understanding of the right level of detail and the right number of (as orthogonal as possible) dimensions most appropriate to the various application cases.

- There is a trade off between the richness of the Context Dimension Tree model and the run-time efficiency of the data-tailoring process. Indeed, when too many dimensions and values are present in the Context Dimension Tree, the run-time specification of the active context may require a long set-up time (e.g., the user might be obliged to set a long list of parameters for the system to identify the current context). The above mentioned considerations on the designers also applies here.

- The definition of the partial view for each context element should be carried out having in mind the strategy that will be used to combine views, and understanding the underlying logic in order not to obtain either the entire global schema or almost empty views.

- CARVE provides a high degree of flexibility with respect to evolutions in the requirements and/or application environment, as far as maintenance is concerned. In fact, should the application environment (and consequently the possible contexts) change, the designer need only adjust the Context Dimension Tree and the affected partial views, letting the tool recompute the new views associated with the possible contexts, with a limited effort from the data tailoring point of view. The context-aware applications on top of such modified views might need to be updated; however, query resilience guarantees that the applications designed with this property in mind would only require a minor intervention.

<table>
<thead>
<tr>
<th>Name</th>
<th># Relations in DB</th>
<th>Avg. # relations per context</th>
<th>Avg. amount of data per context (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pickup your lunch</td>
<td>24</td>
<td>8</td>
<td>35</td>
</tr>
<tr>
<td>Wine</td>
<td>26</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>Mobile student assistant</td>
<td>15</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>Video on demand</td>
<td>5</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>FantaCalcio</td>
<td>10</td>
<td>6</td>
<td>40</td>
</tr>
</tbody>
</table>
The context-aware views derived by means of CARVE lend themselves to two different scenarios: they can be left virtual, as in the classical DB design scenario where several external, virtual user views are used to access an extensional DB, or be materialized, as in the case when a context-aware part of information is materialized on a small, mobile device.

This methodology still hardly lends itself to the highly dynamic scenario of fully pervasive systems, where field-adaptability and situation-awareness are among the main requirements. One step forward is the possibility to compose context-aware views, but much research is still needed to face situations where new, unforeseen contexts may arise and the system has to react without the designer interaction.

9. Concluding remarks

This paper has proposed CARVE, a methodology for the definition of context-aware views over relational databases, exploiting a revised and completed version of a context model and three set-oriented relational operators to compose data views. More specifically, we have discussed the CDT, a context model suitable for expressing the fundamental elements used to tailor data according to the user’s current context, here enriched with constraints which forbid contexts that are not really significant for the given application scenario. The methodology is based on the idea of associating with each element of the context a relevant portion of data (called partial view) which includes the information the user will be interested in, and to automatically combine it with the other aspects characterizing the context, in order to extract (tailor) from the available data the subset of information deemed useful in a given context. To this end, we have augmented and re-defined some relational operators [8] working on sets of relations whose validity goes beyond their application in this context-aware scenario, and which can be used as general-purpose operators for view combination.

Summarizing, the CARVE methodology provides a new way to design external views which, instead of being fixed once and for all for each given user or application, automatically adapt to changes in their situation.

A prototype tool has been developed to support the tailoring process at design time, when partial views are defined, combined and associated with the portions of data that will be provided to the end user at run time. The overall approach has been applied to some case studies, proving it an effective means for context-aware view definition and composition for relational databases.

As future work, we plan to integrate contextual constructs introduced in the paper in an SQL-like query language, to allow one to express queries across contexts.

Acknowledgments

This research has been partially funded by the European Commission, Programme IDEAS ERC, Project 227977-SMS and by the Italian project Industria 2015, Program no. MI10 00091 SENSORI. We wish to thank the anonymous referees for their accurate and acute observations that allowed us to improve the paper.

References


