A graph-based framework for querying and modelling integrated sources

FRANCESCO MERLO

Abstract

The constant growth of the volume of information that organizations have to manage is raising serious problems due to the difficulty in finding the interesting data and lowering the “information noise”. Moreover, data sources are always less homogeneous and need to be integrated in effective ways. The Context-ADDICT framework aims at defining and providing a framework for selecting and integrating the relevant information of the application domain to be delivered to the users on the basis of their current context. This report presents a graph-based conceptual data model to represent the application domain of multiple and integrated data sources, and a related query language that provides a native way to interact with the data model in order to express queries while selecting the relevant portions of the application domain with respect to the context of the user.

Keywords: Graph-based conceptual data model, graph-based query language.
Context-ADDICT is a project on Context-Aware Data Design, Integration, Customization and Tailoring developed at Politecnico di Milano, Dipartimento di Elettronica e Informazione.
An electronic copy of this document is available for download at:
http://poseidon.elet.polimi.it/ca/docs/
A graph-based framework for querying and modelling integrated sources

Francesco Merlo

July 8, 2008

1 Introduction

Nowadays organizations are facing a dramatic growth of information that, if not properly controlled, can lead to a data overload that may cause confusion rather than knowledge, and remarkably reduce the benefits of a rich information system. However, distinguishing useful information from noise, i.e., from all the information not relevant to the specific application, is not a trivial task; the same piece of information can be considered differently, even by the same user, in different situations, or places, – in a single word, in a different context [10,12].

Conceptually, within an information system, users’ needs may depend on two different aspects: the application domain which represents the reality under examination, and the working environment, i.e., the context. This work is part of the Context-ADDICT (Context-Aware Data Design, Integration, Customization and Tailoring) project [2], aimed at defining and providing a framework for selecting and integrating the relevant information of the application domain to be delivered to users on the basis of their current context.

The Context-ADDICT framework adopts a context model, called Context Dimension Tree (CDT), to represent all the situations in which the user may be involved, called contexts; the model is based on a series of context dimensions, each one representing a specific aspect describing the user’s profile (e.g., user’s role or adopted device) or the environment (e.g., location or time). The tailoring process consists of two distinct phases, one performed at design time, the other at run time. At design time, the designer defines a relation between the context model and the application domain: a set of relevant concepts of the application domain (corresponding to a set of relevant data in the database) are associated with each context by means of selection and projection queries. At run time, on the basis of the user’s context, the database server uses the corresponding queries to tailor the suitable view of interest.

Based on these considerations, this work aims at two main objectives: a) the definition of a graph-based conceptual data model that could suitably be used to model the application domain, and b) a related query model to provide a mean to express queries in a native language on such data model.

The application domain is intended as the whole world of interest in which all the concepts related to the users’ activities and needs are represented. Given that in today’s information systems data sources cannot be assumed to have homogeneous representations (i.e., they can be structured, semi-structured or even not structured at all), the need of a conceptual data model that is general enough to represent concepts deriving from different representation is fundamental.

Based on the issues discussed in the previous paragraphs, once the data model has been defined, a suitable query model is required, in order to allow designers to perform data tailoring. The main requirement of the query model is that it should allow the definition of queries in an intensional way and through a direct interaction with the representation of the application domain, rather than in an extensional way as proposed by many approaches available in the literature (cf. [3] for an extensive review of visual query systems).
Figure 1: Graph database models relationships (adapted from [1]). Nodes indicate models and arrows citations.

By achieving these two objectives, the Context-ADDICT framework could be extended with a conceptual data model that can be used to effectively represent the application domain, and a query model that can allow designers to directly interact with such representation to perform data tailoring operations.

The rest of the report is organized as follows. Section 2 presents a review of literature works focusing on the topic of graph-based query languages; Section 3 formally describes the ESDR-Network conceptual data model; Section 4 presents the graph-based query model defined on the proposed data model, while Section 5 concludes the report by drawing conclusions, presenting future work and discussing limitations.

2 Related Work

Several approaches to graph-based query languages have been presented in the literature, as graphs have been associated to databases since a long time, and in particular after the introduction of the object-oriented data models in the middle Eighties. In this context, the structure (schema) of a database is typically represented by means of a graph. As noted by Ullman [26], an important feature of a data model is its associated data language: to deal with the language components, typically schemata in semantic or object-oriented data models are transformed into a conceptual data model, which can be described by an ad-hoc algebra or, for example, the relational model (and the correspondent relational algebra). Consequently, query languages have been specified on the basis of the conceptual data model adopted in each case: this section describes some of the approaches presented in the literature.

An extensive review of graph database models has been presented by Angles and Gutierrez [1]. Figure 1, adapted from [1], shows the main approaches to the definition of graph database models that have been presented in the literature. Some approaches that have been not considered by Angles and Gutierrez have been included in Figure 1. In particular, the approach proposed by Catarci et al. [4], namely Structure Model Hypergraph (SMH), and the derived work by Ursino et al. [25], namely Semantic Distance-Relevance Networks (SDR-Networks). These approaches have been included in the figure in order to provide a clear overview of the theoretical foundations behind the approach presented by this report.

The GDM data model [23] is object-oriented and supports graphs. The query language contextually defined, namely GOQL, is an extension of OQL enriched with constructs to create, manipulate and query objects of type graph, path and edge. Another feature of GOQL is the capability of querying sequences and paths: in addition to the sequence operators of OQL, GOQL uses the temporal operators $next$, $until$ and $connected$ for queries involving the relative ordering of sequence elements. GOQL queries are translated into an operator-based language, namely O-Algebra, which has been defined by the authors.
themselves [13].

The GOOM data model [7, 8] has been proposed as an abstract data model, the corresponding data structure and a formal graph-based algebra (called G-algebra) that operates on the structured data model. The proposed data model supports the standard features of the object-oriented model (association, aggregation and inheritance), inherently ensuring referential integrity.

GOOD [11,17] is a project which addresses explicitly the issue of providing a graphical user interface in addition to the data model and the data manipulation language. It is typical that in the graph-based database context user interfaces are required to allow users to express queries visually by graphs, which should be built on the basis of the components of the schema graph. In GOOD the function of the query graph is abstracted in the concept of pattern (which is also used as primitive for other data manipulation operations). The pattern structure specifies which portions of the database are to be retrieved by the query, which is thus performed by means of pattern-matching mechanism. These are formalized by means of the extended Tarski-algebra [24], which is proved to be as expressive as the relational algebra [14]. GOOD has also been implemented as a functioning system at the Indiana University [22]. The main limitation of the GOOD approach is that edges carry no information but their labels: edges with semantics need to be modeled as nodes, and therefore loose their natural status of edges; moreover, in the GOOD model only nodes are associated in a natural way to properties and values [17].

G-Log [16] is a graph-based, declarative, nondeterministic complete query language which combines the expressive power of logic, the modeling power of objects with identity and the representation power of graphs. The data model of G-Log uses directed labeled graphs to represent database schemata, and is (up to some minor details) the same as that of GOOD. The main difference between the two approaches is that G-Log is a declarative and nondeterministic language, while GOOD is imperative and deterministic.

While the proposed approaches analyzed above are all suitably formally defined, present interesting features and are at least as expressive as the relational algebra, none of them allow query definition in a suitable way as needed in the Context-ADDICT framework. Although G-Log [16] allow the user to visually define queries, it is required that a sequence of rules describing the desired results is defined by the user. What is needed is instead a way to point out the relevant areas (i.e., nodes and edges) of the global conceptual schema directly interacting with a graphical representation of the SDR-Network model. The next section presents an approach (namely, SMH) that allow such a direct visual interaction with the schema representation.

2.1 Structure Model Hypergraph (SMH)

This section presents a more detailed description of an approach to the definition of a visual graph-based query language. Section 2.1.1 describes the Structure Modeling Hypergraph data model, while Section 2.1.2 illustrates how it can be mapped to the relational data model; finally, Section 2.1.3 presents how the SMH data model can be queried, with particular attention to the Relational SMH data model.

2.1.1 SMH Data Model

Structure Modeling Hypergraphs (SMH) have been defined by Catarci et al. in [4] and subsequently extended in [5, 6]. Generally speaking, an SMH consists of a set of labeled nodes, representing distinguishable concepts and their properties, and a set of (directed or undirected) hyperedges used to describe their structure. Formally, a Modeling Hypergraph $MH$ is a 6-tuple:

$$MH = \langle \mathcal{N}, \mathcal{A}, \mathcal{H}_U, \mathcal{H}_D, \mathcal{L}, \lambda \rangle,$$

where:

- $\mathcal{N} = \mathcal{N}_I \cup \mathcal{N}_U$ is a (possibly empty) set of nodes, $\mathcal{L}$ is a set of labels and $\lambda$ is a function from $\mathcal{N}$ to $\mathcal{L}$ such as all nodes are labeled and the nodes in $\mathcal{N}_I$ are univocally identified by their labels (i.e. the restriction $\lambda|_{\mathcal{N}_I}$ is injective and the ranges of $\lambda|_{\mathcal{N}_I}$ and $\lambda|_{\mathcal{N}_U}$ are disjoint);

- $\mathcal{A}$ is a set of arcs (directed edges), and an arc $A$ is an ordered pair $A = \langle i, j \rangle$, where $i, j \in \mathcal{N}$;
Figure 2: A SMH representing information about persons.

- $\mathcal{H}_U$ is a set of (possibly undirected) hyperedges, and a hyperedge is either:
  - a non-empty (not necessarily proper) subset of $\mathcal{N}$ (simple hyperedge), or
  - a non-empty multiset of hyperedges;

- $\mathcal{H}_D$ is a set of hyperarcs (directed hyperedges) and an hyperarc $H_D$ is an ordered pair $H_D = (i, h)$ for which $i \in \mathcal{N}$ and $h$ is an hyperedge.

Furthermore, an $MH = (\mathcal{N}, \mathcal{A}, \mathcal{H}_U, \mathcal{H}_E, \mathcal{L}, \lambda)$ is an SMH if all the following conditions hold:

- either $\mathcal{N} = \emptyset$ or $\mathcal{N}_I \neq \emptyset$;
- an arc $A \in \mathcal{A}$ is an ordered pair $A = (i, j)$, where $i \in \mathcal{N}_U$ and $j \in \mathcal{N}_I$;
- $\mathcal{H}_U$ is partitioned in three sets: $\mathcal{H}_S$ (structure hyperedges), $\mathcal{H}_E$ (extension hyperedges) and $\mathcal{H}_G$ (group hyperedges), having the following properties:
  - structure hyperedges can be used for representing the notion of object structure as a set of properties (each property being represented by a node of the hyperedge). Each property is linked to a target, which may be an individual value (i.e., an identifiable node) or a set of values (i.e., a group hyperedge), depending on the nature of the property;
  - extension hyperedges can be used to represent collections of individuals sharing common features. They consist of either nodes when representing data model exhibiting object identity or structure hyperedges otherwise (e.g., in the relational model a structure hyperedge represents a tuple, and an extension hyperedge a relation). The existence of exactly one incoming hyperarc can be referred to as extension ownership uniqueness;
  - Group hyperedges can represent sets of individuals different from extensions (e.g., the target of a multi-valued property);
- each node $i \in \mathcal{N}_I$ is connected to at most one structure hyperedge and to at most one extension hyperedge; these conditions can be referred to as structure uniqueness and extension uniqueness.

Figure 2 illustrates by means of a simple example an admissible SMH, together with a possible semantics, without any concern about the data model. It is worth noting that nodes in $\mathcal{N}_I$ (filled in black) are used to represent the distinguishable objects in our reality of interest, whereas nodes in $\mathcal{N}_U$ (filled with an horizontal texture) represent the properties that describe them. As for the hyperedges, those in $\mathcal{H}_S$ (ovals with gray texture) are used to associate to nodes in $\mathcal{N}_I$ the description of their
structure in terms of a set of properties, while hyperedges in $H_E$ (ovals with white texture) represent the extension of nodes in $N_I$. The link between nodes and hyperedges is realized in both cases by an hyperarc $h_D \in H_D$ (all the hyperarcs are represented by arrows). Finally, an arc connects each node belonging to a structure hyperedge to the value of the associated property.

2.1.2 Mapping SMH to the Relational Data Model

The relational model, proposed by Codd in [9], represents data in a database as a collection of relations. Informally, each relation resembles a table, and each row in the table represents a collection of related data values. The following definitions specialize the SMH constituents to represent a relational database.

An $SMH = (N, A, H_U, H_D, L, \lambda)$ is a Relational SMH (RSMH) if the following conditions hold:

- the set of identifiable nodes $N_I$ is partitioned into three subsets $N_D, N_V$ and $N_{RN}$;
- either $N = \emptyset$ or $N_{RN} \neq \emptyset$;
- the set of arcs $A$ is partitioned into two subsets $A_{Dom}$ and $A_{Val}$, where $A_{Dom} \subseteq N_U \times N_D$, and $A_{Val} \subseteq N_U \times N_V$;
- the set of structure hyperedges $H_S$ is partitioned into two subsets $H_{Sch}$ and $H_{Tuple}$:
  - hyperedges in $H_{Sch}$ are used to represent a relational schemata, each attribute being represented by a node of the hyperedge, linked to a node in $N_D$ representing the attribute domain;
  - hyperedges in $H_{Tuple}$ are used to represent tuples, each attribute being given by a node of the hyperedge, linked to a node in $N_V$ representing an attribute value;
- the set of hyperarcs $H_D$ is partitioned into three subsets $H_{Attr}, H_{Inst}$ and $H_{Val}$, where $H_{Attr} \subseteq N_{RN} \times H_{Sch}, H_{Inst} \subseteq N_{RN} \times H_{Ext}$ and $H_{Val} \subseteq N_D \times H_{Set}$. It also holds that for each $i \in N_D$ there exists exactly one outgoing hyperarc in $H_{Val}$, and for each $i \in N_{RN}$ there exists exactly one outgoing hyperarc in $H_{Inst}$;
- $H_G = \emptyset$;
- the set of extension hyperedges $H_E$ is partitioned into two subsets $H_{Ext}$ and $H_{Set}$:
  - hyperedges in $H_{Set}$ are used to represent attribute domains;
  - hyperedges in $H_{Ext}$ are used to represent relations. It is guaranteed that hyperedges in $H_{Tuple}$ corresponding to the same schema actually share the same structure, and that values of tuples are taken from the domains of the corresponding schema; tuple uniqueness is also guaranteed.

Nodes, labels, arcs, hyperarcs and hyperedges are associated to the concepts of the relational model as shown in table 1, while Figure 3 shows an example of a relational database represented by means of a RSMH.

2.1.3 Querying SMHs

The interaction with an SMH is carried on by simple primitives acting on the status of SMH elements. During an interaction session the entire SMH is available and its elements may switch their status from ON to OFF and vice versa. At the beginning of the session, all the elements are in the OFF status, and may be set to ON by graphical selection. The underlying idea is that the SMH is partitioned into two distinct parts: the ON part and the OFF part; it is required that the ON part conforms to all the general constraints after each interaction step.

Switching an element status between ON and OFF requires the collateral analysis of the status of other elements belonging to the SMH to preserve the consistency of the ON and OFF parts. Since these are disjoint, any adding of an element to one of them implies its removal from the other; furthermore, it is intuitive that switching an element to the OFF status should imply a propagation of the operation to
Figure 3: A relational database represented by means of a RSMH.
the sub-hypergraph rooted at such element: however, this operation must be performed by maintaining ON those elements of a sub-hypergraph that are shared by other sub-hypergraphs rooted at other ON elements.

The query formulation is composed by the following sequence of operations:

**Query activation** In this phase, the SMH query system automatically creates a temporary identifiable node, leaving its structure and extension empty.

**Structure definition** In this phase, the user selects by means of ON operations a number of unidentifiable nodes belonging to structure hyperedges. The system automatically adds copies of those unidentifiable nodes to the structure hyperedge owned by the result, together with their outgoing arcs.

**Extension definition** In this phase, the user selects by means of ON operations a number of components of extension hyperedges. The system automatically inserts new elements in the result extension, suitably composing selected components, according to the conditions imposed to the SMH by the specific data model.

The sequence of operations described above will be presented in detail with respect to the RSMH data model discussed in the previous section.

The main issue to be faced is the formulation of single-operators queries, while complex queries may be formulated in an incremental way as a series of single operator queries, since the property of closure under queries holds for the relational data model (see [26]).

Considering a generic relational algebra expression \( E \), the corresponding RSMH representation is denoted as \( r = M(E) \); as already stated, the following notation will be used: a relation is represented by a triple \( r = (rn, h_{Attr}, h_{Inst}) \), where \( rn \in \mathcal{N}_{RN} \), \( h_{Attr} \in \mathcal{H}_{Attr} \), \( h_{Inst} \in \mathcal{H}_{Inst} \), with \( h_{Attr} \) and \( h_{Inst} \) originating from \( rn \).

Table 2 shows the specification of the actions described for the formulation of a query in the cases of a simple algebra expression and of composed algebra expressions containing union, difference, projection, selection and product operators.

Figure 4 illustrates by means of a visual example the three phases of a query on a RSMH. The database is the same of that in Figure 3, and represents the relation between Employees and Companies. The query activation phase creates the node myQuery with the associated \( H_{Sq} \) and \( h_{Eq} \) empty hyperedges. The user then selects a node belonging to the structure hyperedges of the RSMH (structure definition phase): in this case, the node City from the structure hyperedge of Companies has been selected: the system therefore adds a copy of the selected node to the structure hyperedge \( H_{Sq} \) of the node myQuery. The extension definition phase is in this case automatically performed by the system, which populates the extension hyperedge \( h_{Eq} \) of myQuery with copies of the nodes matching the user selection of node City from the structure hyperedge of Companies. In this case only two tuples are inserted, since the RSMH contains two instances of Companies. The query can be formulated in SQL as:

```sql
SELECT c.city 
FROM companies AS c
```

### 3 ESDR Network Model

This section presents the data model defined as main objective of this work. In particular, Section 3.1 presents the SDR-Network conceptual data model, on which the ESDR-Network is based on. The Extended SDR-Network conceptual data model is presented in Section 3.2.

#### 3.1 SDR Network Conceptual Data Model

The SDR-Network data model has been defined by Ursino et al. in [25] and subsequently refined and used as a conceptual data model for integration purposes in [15,18–21].
Figure 4: Querying a RSMH. a) Query activation; b) Schema definition; c) Extension definition.
A SDR-Network \( \text{Net}(DS) \) representing a data source \( DS \) is formally defined as a rooted labeled graph:

\[
\text{Net}(DS) = \langle \mathcal{N}(DS), \mathcal{E}(DS) \rangle = \langle \mathcal{N}_A(DS) \cup \mathcal{N}_C(DS), \mathcal{E}(DS) \rangle
\]

where \( \mathcal{N}(DS) \) is the set of nodes and \( \mathcal{E}(DS) \) the set of edges.

Each node \( N \in \mathcal{N}(DS) \) represents a concept of \( DS \), is identified by the name of the corresponding concept and represents all instances of that concept in \( DS \). Node \( N \) can be either atomic or complex. \( N \) is defined as atomic (\( N \in \mathcal{N}_A(DS) \)) if it does not have outgoing edges, complex otherwise (\( N \in \mathcal{N}_C(DS) \)).

Each edge \( E \in \mathcal{E}(DS) \) represents a relationship between two concepts of \( DS \). More specifically, if an edge \( E \) insists from the node \( S \) to the node \( T \) and is labeled \( L_{ST} \), is denoted by the triple:

\[
E = \langle S, T, L_{ST} \rangle,
\]

and indicates that the concept represented by \( S \) (source node) is semantically related to the concept represented by \( T \) (target node).

The label \( L_{ST} \) is in turn defined as the pair of coefficients:

\[
L_{ST} = \langle d_{ST}, r_{ST} \rangle,
\]

where both \( d_{ST} \) and \( r_{ST} \) belong to the real interval \([0, 1]\). In particular, \( d_{ST} \) is called semantic distance coefficient and is used to indicate how much the concept associated with node \( T \) is related to the concept expressed by \( S \); this depends on the capability of the concept associated with \( T \) to characterize the concept denoted by \( S \). The coefficient \( r_{ST} \) is called semantic relevance coefficient and represents the fraction of instances of the concept denoted by \( S \) whose complete definition requires at least one instance of the concept represented by \( T \).

![Figure 5: The SDR-Network representing a clothes shop.](image)

An example of SDR-Network is shown in Figure 5; it corresponds to a data source describing a clothes shop. In the figure, in order to simplify the layout, a gray node having the name \( x \) is used to indicate that the edge incident onto \( x \) must be considered incident onto the corresponding white node having the same name. SDR-Network nodes such as \( Clothes, Supplier \) or \( Model \) represent the involved concepts. The edge \( \langle Clothes, Discount, [1, 0.25] \rangle \) denotes the existence of a relationship between \( Clothes \) and \( Discount \); in
particular, it indicates that 25% of clothes have a discount. The other edges have analogous semantics. In the figure, the coefficient associated with each node indicates the number of instances of the concept the node represents; moreover, the number in parentheses associated with each edge is the so-called number of relevant instances associated with the edge; this represents the number of instances of the source node which require at least one instance of the target node for their complete definition.

3.2 Extended SDR Network Conceptual Data Model

This section describes the ESDR-Network Network conceptual data model. The SDR-Network conceptual data model can be used to represent a generic data source (e.g., based on the relational data model); however, once the SDR-Network is derived from the data source, it is not possible to draw correspondences with the original data source, since some fundamental details are left out (e.g., the specialization relationships between entities in a data source represented by means of the ER data model). Since the Context-ADDICT framework requires that the queries expressed on the application domain have to be suitably translated to the original data sources, the SDR-Network data model must be enriched to support such characteristic.

An ESDR-Network $\text{ENet}(DS)$ representing a data source $DS$ is formally defined as a rooted labeled graph:

$$\text{ENet}(DS) = \langle N(DS), E(DS) \rangle = \langle N_A(DS) \cup N_C(DS), E(DS) \rangle$$

where $N(DS)$ is the set of nodes and $E(DS)$ the set of edges.

Each node $N \in N(DS)$ represents a concept of $DS$, is identified by a label $\langle \text{label} \rangle$ (the name of the corresponding concept) and represents all instances of that concept in $DS$. Node $N$ can be either atomic or complex. $N$ is defined as atomic ($N \in N_A(DS)$) if it does not have outgoing edges, complex otherwise ($N \in N_C(DS)$).

Each edge $E \in E(DS)$ represents a relationship between two concepts of $DS$, as well as in the SDR-Network model. More specifically, an edge $E$ insisting from the node $S$ to the node $T$ and labeled $L_{ST}$ is denoted by the triple:

$$E = \langle S, T, L_{ST} \rangle,$$

and indicates that the concept represented by $S$ (source node) is semantically related to the concept represented by $T$ (target node). The label $L_{ST}$ is in turn defined as the following triple:

$$L_{ST} = \langle \text{edge\_name}, d_{ST}, r_{ST} \rangle,$$

where:

- $\text{edge\_name}$ is the textual label associated with the edge;
- $d_{ST}$ is the semantic distance coefficient, as in the SDR-Network model;
- $r_{ST}$ is the semantic relevance coefficient, as in the SDR-Network model;

Conversely to what defined by the SDR-Network conceptual data model, in the ESDR-Network model two nodes can be connected by more than one edge, in order to allow the representation of different but coexisting relations between the two nodes.

In order to allow a precise description of the concepts that have to be represented through the ESDR-Network conceptual model, two particular edge types are defined:

**Key edge** This edge is defined to identify the attribute that is used as key for a given concept. It is required that the source node $S$ of the edge is a complex node, that is $S \in N_C$, and the target node $T$ of the edge is an atomic node, that is, $T \in N_A$. When the definition of a key is required for a complex node, one or more atomic nodes can be linked by means of key edges, in order to represent simple or composed keys. A key edge is identified by the label $\text{IdentifiedBy}$.

Figure 6a shows an example of simple key, since the node $\text{Person}$ is identified by a single attribute, $ID$. Figure 6b shows an example of a composed key: in this case, the node $\text{Person}$ is identified by a set of its attributes, namely $\text{Name}$, $\text{Surname}$ and $\text{Birth Date}$. 
Generalization edge This type of edge is defined in order to provide explicit support to the generalization feature of the entity-relationship model. A generalization edge from node S to node T expresses the concept that node S “is a” T, that is, T is a generalization of the concept represented by S, or S is a specialization of the concept represented by T. A generalization edge is identified by the label IsA.

Figure 7 shows an example of key and generalization edges: a Person is identified by an ID and has a set of attributes (Name, Surname and Birth Date). An Employee is a specialization of the concept of Person, and is consequently linked to the node Person by an IsA edge; the Employee has all the attributes of the node Person, but also others that are related only to the concept of Employee, such as Department, Salary, Phone Number and E-mail.

An example of ESDR-Network is presented in Figure 8; it describes an estate agency. In the figure, in order to simplify the layout, the semantic distance and relevance indexes of each edge have not been shown. It is worth noting that several characteristics of the ESDR-Network conceptual data model are used in the figure. In particular:

- All the edges have been labeled with a distinctive name.
- More than one edge can insist between the same two nodes. For example, two different edges exists between the nodes Agency and Agent.
- Nodes can be put in hierarchical relationship by means of generalization edges. For example, the Person node is the root of a hierarchy that involves also the Client and the Agent nodes. The former is in turn specialized by the type of client of the agency, namely Owner, Buyer or Renter, depending on the type of relationship that ties each client to the agency.
- Key edges are used to link concepts to the respective identification edges. For example, a Person is identified by his/her SSN.

4 ESDR Network Query Model

This section provides a detailed description of the query model that has been defined over an ESDR-Network. In particular, Section 4.1 presents the primitive operators that allow the visual interaction with
4.1 Interaction Primitives

This subsection introduces the interaction primitives that allow the visual interaction with an ESDR-Network.

The interaction primitives basically deal with the status management of each element composing an ESDR-Network, and allow the user to switch the elements’ status from ACTIVE to INACTIVE and vice versa. The initial default status of each element of an ESDR-Network is INACTIVE, and can be switched to ACTIVE by means of visual interaction (basically, by selecting the element). The selection of an elements whose status is ACTIVE causes its deactivation, that is, the switch of its status to INACTIVE. Given a ESDR-Network \( N \), the elements whose status is switched to ACTIVE are added to an active layer \( ESDR_{ACTIVE} \). Whenever an element’s status is switched to INACTIVE, the element is removed from the active layer.

The management of status switch is performed through the application of status functions, that allow the switch of an element’s status preserving the general constraints; namely, the function \( ACTIVATE(e) \) switches the status of the generic element \( e \) from INACTIVE to ACTIVE, while the function \( DEACTIVATE(e) \) switches the status from ACTIVE to INACTIVE. The following rules provide a description of the application of the \( ACTIVATE(e) \) function to the elements of an ESDR-Network. Please note that the status functions are idempotent, i.e., the application of the \( ACTIVATE() \) (respectively, \( DEACTIVATE() \)) function to an element whose status is already ACTIVE (respectively, INACTIVE) has no effects.

Complex nodes. If \( e \) is a complex node, i.e. \( e \in N_C(DS) \), the switch of its status causes the activation of all its connected atomic nodes. Algorithm 1 shows in pseudo-code style the operations that are performed when activating a complex node \( e \): besides the status switch of \( e \), each atomic node \( N_i \) connected to \( e \) through an edge \( E(e, N_i) \) is added to the active layer \( ESDR_{ACTIVE} \).

an ESDR-Network, Section 4.2 provides a description of the relational algebra operators applied to the ESDR-Network conceptual data model, while Section 4.3 describes the ESDR-Network query specification procedure.
Algorithm 1 Activation of a complex node.

Require: $e \in \mathcal{N}_C(\mathcal{DS})$
Ensure: $e \in ESDR_{ACTIVE}; \forall N_i \in \mathcal{N}_A(\mathcal{DS}) \mid \exists E(e, N_i) \in \mathcal{E}(\mathcal{DS}), N_i \in \mathcal{N}_{ACTIVE} \land E(e, N_i) \in ESDR_{ACTIVE}$

procedure ACTIVATE($e$)
    $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup \{e\}$;
    for all $N_i \in \mathcal{N}_A(\mathcal{DS})$ such that exists $E(e, N_i) \in \mathcal{E}(\mathcal{DS})$ do
        $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup \{N_i, E(e, N_i)\}$;
    end for
end procedure

Atomic nodes. If $e$ is an atomic node, that is $e \in \mathcal{N}_A(\mathcal{DS})$, the switch of its status causes the activation of its parent node and the edge connecting the two nodes. If the atomic node is connected to more than one complex node, all the complex nodes are added to the active layer. Algorithm 2 shows that besides the atomic node $e$, all the complex nodes $N_i$ and the connection edges $E(N_i, e)$ are added to the active layer $ESDR_{ACTIVE}$.

Algorithm 2 Activation of an atomic node.

Require: $e \in \mathcal{N}_A(\mathcal{DS})$
Ensure: $e \in ESDR_{ACTIVE}; \forall N_i \in \mathcal{N}_C(\mathcal{DS}) \mid \exists E(N_i, e) \in \mathcal{E}(\mathcal{DS}), N_i \in \mathcal{N}_{ACTIVE} \land E(N_i, e) \in ESDR_{ACTIVE}$

procedure ACTIVATE($e$)
    $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup \{e\}$;
    for all $N_i \in \mathcal{N}_C(\mathcal{DS})$ such that exists $E(N_i, e) \in \mathcal{E}(\mathcal{DS})$ do
        $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup \{N_i, E(N_i, e)\}$;
    end for
end procedure

Edges. If $e$ is an edge $E(N_i, N_j) \in \mathcal{E}(\mathcal{DS})$ insisting between two nodes $N_i$ and $N_j$, the switch of its status implies no other operations. Algorithm 3 shows that only the edge $e$ is added to the active layer $ESDR_{ACTIVE}$.

Algorithm 3 Activation of an edge.

Require: $e \in \mathcal{E}(\mathcal{DS})$
Ensure: $e \in ESDR_{ACTIVE}$

procedure ACTIVATE($e$)
    $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup \{e\}$;
end procedure

4.2 ESDR-Network Relational Algebra Operators

This section provides a description of the operators that can be used to specify a query over an ESDR-Network. The relational algebra [9] is the query model adopted at the basis of the presented operators. For each operator, a relational algebra definition is provided as reference, along with the corresponding definition in the ESDR-Network query model. In the following paragraphs, the symbol $R(X)$ indicates a relation schema $R$ defined on a set of attributes $X = \{A_1, A_2, \ldots, A_n\}$, while the symbol $r(X)$ indicates an instance on the schema $R(X)$. 
It is worth noting that the presented operators are defined only in the active layer of an ESDR-Network. That is, the edges that are used to identify the operators should not be intended to be particular subtypes of the ESDR-Network conceptual model edges defined in Section 3.2.

4.2.1 Rename

Relational algebra definition. The rename operator is defined in the relational algebra on a relation \( r(X) \) defined on an attribute set \( X \), and another attribute set \( Y \), as \( \varrho_{B_1, B_2, \ldots, B_n}(A_1, A_2, \ldots, A_n)(r) \), where \( A_1, A_2, \ldots, A_n \) and \( B_1, B_2, \ldots, B_n \) are orderings for the elements of the attribute sets \( X \) and \( Y \), respectively. The result of the operator is a set of tuples containing a tuple \( t' \) for each tuple \( t \) in \( r \); each tuple \( t' \) is defined on the attribute set \( Y \) such as \( t'[B_i] = t[A_i] \) for \( i = 1, \ldots, n \).

ESDR-Network definition. In the ESDR-Network model, the rename operator corresponds to the assignment, for each atomic node belonging to set \( X \), of the corresponding name specified in the set \( Y \). It is assumed that such specification has to be performed only when the new name is different from the existing one, that is, when \( B_i \neq A_i \), for \( i = 1, \ldots, n \).

Algorithm 4 Rename.

Require: \( N_i \in \mathcal{N}_A(DS) \land \text{label}(N_i) = (A_i) \) for \( i = 1, \ldots, n \); \( X = \{A_1, \ldots, A_n\} \); \( Y = \{B_1, \ldots, B_n\} \)

Ensure: \( \text{label}(N_i) = (B_i) \) for \( i = 1, \ldots, n \)

procedure \text{RENAME}(X, Y)
  for all \( N_i \in \mathcal{N}_A(DS) \mid \text{label}(N_i) = A_i \) do
    if \( B_i \neq A_i \) then
      \text{label}(N_i) = (B_i);
    end if
  end for
end procedure

Figure 9: Rename.

Figure 9 shows an example of application of the rename operator to a complex node \( N \), defined on the attribute set \( X = \{A_1, A_2, A_3, A_4\} \). The rename set \( Y = \{B_1, A_2, B_2, A_4\} \) has been applied, and, as shown by the figure, attributes \( A_1 \) and \( A_3 \) have been respectively renamed as \( B_1 \) and \( B_2 \). Since the new names of the other attributes are equal to the existing ones, they have been kept unchanged. Algorithm 4 formally describes the Rename operator.

Renaming is potentially applicable also to complex nodes, in order to allow the expression of queries involving variables (for example, if the same complex node has to be referred to more than once in the same query). However, it has been chosen not to provide support to such operator at the moment. Such capability could be provided by replicating parts of the ESDR-Network in order to allow the visual interaction with the two (or more) copies of the same complex node.

Figure 10 shows an example of a possible query requiring the renaming of the complex node \( N \). Considering the query “retrieve all the elements in \( N \) having the same values for attribute \( A_1 \) but different values of attribute \( A_2 \)”, it is required to address twice the complex node \( N \). As shown in Figure 10, the
node has been copied and properly renamed in order to avoid name clashes; it is now part of the ESDR-Network. Once such operation is performed, it is possible to interact with the two nodes as if they were completely distinct.

### 4.2.2 Set Operators

This section provides a description of the set operators *union*, *intersection* and *difference* contextualized to the ESDR-Network model. For each operator the relational algebra definition is reported, along with the corresponding formal definition for the ESDR-Network model.

**Nodes compatibility** The application of the set operators requires the involved complex nodes to be compatible. Two complex nodes $N$ and $M$ are defined as compatible if and only if the domains of the elements of the respective attribute sets are compatible. The correspondence between the elements of the attribute sets is based on attribute names; i.e., the elements of the two attribute sets should be put in bijective correspondence by considering their (alphabetically ordered) names. Attributes can be eventually renamed to make attributes sets comparable.

**Union**

**Relational algebra definition.** The union operator is defined in the relational algebra as $r_1 \cup r_2$ on two compatible relations $r_1$ and $r_2$. The result of the operator is the set of tuples belonging to $r_1$, to $r_2$ or both relations.

**ESDR-Network definition.** Considering the ESDR-Network model, the union operator corresponds to the connection of the desired complex nodes through a **UNION** arc, provided that they are compatible.

**Algorithm 5** Union.

**Require:** $N_1, N_2 \in \mathcal{N}_C(DS)$

**Ensure:** $N_1, N_2, E(N_1, N_2, \langle \text{UNION}, \text{result\_name} \rangle) \in ESDR_{ACTIVE}$

**procedure** Union($N_1, N_2$)

- $activate(N_1)$;
- $activate(N_2)$;
- $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup E(N_1, N_2, \langle \text{UNION}, \text{result\_name} \rangle)$

**end procedure**
An example of application of the union operator is shown in Figure 11. In this case, nodes $N$ and $M$ are not compatible, since the elements of their attribute sets cannot be put in one-to-one correspondence; however, the rename operator allows to make the nodes compatible by correctly renaming the attributes that do not have correspondence (this operation actually corresponds to the definition of the attribute set of the result). The formal definition of the Union operator is reported in Algorithm 5.

The resulting set of tuples will be named as specified by the label of the UNION arc: for example, in Figure 11, the result of the application of the operator will be named $K$, and will be defined on the attribute set $\{A_1, A_2, X_1, X_2\}$.

It is worth noting that the UNION arc is directed even if the operator’s semantic would not require so, since the operator is commutative (i.e., $r_1 \cup r_2 = r_2 \cup r_1$). The directedness of the arc is used only to preserve the correctness of the subgraph defined to apply the operator (since it is required to be a rooted connected subgraph).

**Intersection**

**Relational algebra definition.** The intersection operator is defined in the relational algebra as $r_1 \cap r_2$ on two compatible relations $r_1$ and $r_2$. The result of the operator is the set of tuples belonging to $r_1$ and to $r_2$.

**ESDR-Network definition.** In the ESDR-Network model, the intersection operator corresponds to the connection of the desired complex nodes through an INTERSECT arc, provided that they are compatible.

---

**Algorithm 6 Intersection.**

Require: $N_1, N_2 \in \mathcal{N}_C(\text{DS})$

Ensure: $N_1, N_2, E(N_1, N_2, \langle \text{INTERSECT}, \text{result\_name} \rangle) \in ESDR_{ACTIVE}$

procedure INTERSECT($N_1, N_2$)

\[ \text{activate}(N_1); \]

\[ \text{activate}(N_2); \]

\[ ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup E(N_1, N_2, \langle \text{INTERSECT}, \text{result\_name} \rangle) \]

end procedure

---

![Figure 11: Union.](image)

![Figure 12: Intersection.](image)
An example of application of the intersection operator is shown in Figure 12. In this case, nodes $N$ and $M$ are not compatible, since the elements of their attribute sets cannot be put in one-to-one correspondence: however, the rename operator allows to make the nodes compatible by correctly renaming the attributes that do not have correspondence (this operation actually corresponds to the definition of the attribute set of the result). Algorithm 6 formally describes the Intersect operator.

The resulting set of tuples will be named as specified by the label of the INTERSECT arc: for example, in Figure 11, the result of the application of the operator will be named $K$, and will be defined on the attribute set $\{A_1, X_1, A_3, X_2\}$.

It is worth noting that the INTERSECT arc is directed even if the operator’s semantic would not require so, since the operator is commutative (i.e., $r_1 \cap r_2 = r_2 \cap r_1$). The directedness of the arc is used only to preserve the correctness of the subgraph defined to apply the operator (since it is required to be a rooted connected subgraph).

**Difference**

**Relational algebra definition.** The difference operator is defined in the relational algebra as $r_1 - r_2$ on two compatible relations $r_1$ and $r_2$. The result of the operator is the set of tuples belonging to $r_1$ but not to $r_2$.

**ESDR-Network definition.** In the ESDR-Network model, the difference operator corresponds to the connection of the desired complex nodes through a MINUS edge, provided that they are compatible.

**Algorithm 7 Difference.**

```plaintext
Require: $N_1, N_2 \in N_C(DS)$
Ensure: $N_1, N_2, E(N_1, N_2, (\text{MINUS}, \text{result\_name})) \in ESDR_{ACTIVE}$

procedure DIFFERENCE($N_1, N_2$)
    activate($N_1$);
    activate($N_2$);
    $ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup E(N_1, N_2, (\text{MINUS}, \text{result\_name}))$
end procedure
```

Figure 13: Difference.

An example of application of the difference operator is shown in Figure 13. In this case, nodes $N$ and $M$ are not compatible, since the elements of their attribute sets cannot be put in one-to-one correspondence: however, the rename operator allows to make the nodes compatible by correctly renaming the attributes that do not have correspondence (this operation actually corresponds to the definition of the attribute set of the result). The Difference operator is formally described in Algorithm 7.

The resulting set of tuples will be named as specified by the label of the MINUS arc: for example, in Figure 11, the result of the application of the operator will be named $K$, and will be defined on the attribute set $\{X_1, A_2, X_2, A_4\}$.

The order of the nodes selected is made explicit by the direction of the MINUS arc: since the arrow goes from node $N$ to node $M$, the operation is $N - M$. 

17
4.2.3 Projection

Relational algebra definition. The projection operator is defined in the relational algebra as $\Pi_Y(r)$ on a relation $r(X)$, where $Y$ is a subset of $X$. The result of the operator is the set of tuples on $Y$ obtained from the tuples of $r$ considering only the values on $Y$:

$$\Pi_Y(r) = \{ t[Y] \mid t \in r \}.$$  

ESDR-Network definition. Considering the ESDR-Network model, the projection operator corresponds to the selection of atomic nodes of a complex node. That is, given a node $N$ and its set of atomic nodes $X = \{ A_1, \ldots, A_n \}$, the projection on the subset $Y$ of $X$ is performed by selecting all the atomic nodes belonging to $Y$.

Algorithm 8 Projection.

Require: $Y \subseteq X = \{ A_1, \ldots, A_n \}$; $\forall A_i \in Y, A_i \in N(\text{DS})$ 

Ensure: $\forall A_i \in Y, A_i \in \text{ESDR}_{\text{ACTIVE}}$

procedure Project($Y$)  
for all $A_i \in Y$ do
activate($A_i$);
end for
end procedure

Figure 14: Projection.

The application of the projection operator requires that a complex node is selected, in order to provide the attribute set $X$ that has to be considered when defining the projection subset $Y$. As a consequence, the selection of a generic atomic node causes the selection of the corresponding complex node, if it is not already selected. The direct selection of a complex node causes instead the selection of all of its atomic nodes.

The projection operator alone can be applied to one complex node at a time: thus, it is not allowed to select one (or more) atomic nodes from different complex nodes. As for the relational algebra operator, the selection of the primary key atomic node is not imposed.

Figure 14a shows an example of projection for a node $N$ with four attributes (i.e., $X = \{ A_1, A_2, A_3, A_4 \}$) on the attribute set $Y = \{ A_2, A_3 \}$, while Figure 14b shows the effect of the direct selection of all the attributes of the complex node $N$. Algorithm 8 provides the formal description of the Project operator.

4.2.4 Selection

Relational algebra definition. The selection operator is defined in the relational algebra as $\sigma_F(r)$ on a relation $r(X)$, where $F$ is a propositional formula obtained by combining with the logical operators $\lor$, $\land$ and $\neg$ atomic conditions of the form $A \theta B$ or $A \theta c$, where:

- $\theta$ is a comparison operator ($=, \neq, >, <, \geq, \leq$);
• $A$ and $B$ are attributes in $X$ on whose values the comparison $\theta$ is meaningful;
• $c$ is a constant.

The result of the operator is a subset of tuples of $r$ containing those tuples for which the propositional formula $F$ is true:

$$
\sigma_{A \theta B}(r) = \{ t \mid t \in r, t[A] \theta t[B] \}
$$

$$
\sigma_{A \theta c}(r) = \{ t \mid t \in r, t[A] \theta c \}
$$

**ESDR-Network definition.** In the ESDR-Network model, the selection operator corresponds to the specification of the selection condition $F$ on the atomic nodes of one or more complex node(s). That is, given a node $N$ and its set of atomic nodes $X = \{A_1, \ldots, A_n\}$, the specification of the selection condition $F$ is performed by

- normalizing the propositional formula $F$ in the disjunctive normal form and
- specifying the atomic conditions of each component of the normalized form of the selection condition $F$ to the related attribute(s) of node $N$.

---

**Algorithm 9 Selection.**

Require: $N \in \mathcal{N}_C(DS)$; $F$ expressed in disjunctive normal form on the attributes of $N$

Ensure: $N \in ESDR_{ACTIVE}$

procedure \textsc{Select($N,F$)}

$ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup N$;

for all selection layers $L_j \in F$ do

for all atomic conditions $F_j \in L_j$ do

if $F_j$ type of $A \theta B$ then

$SC = SC \cup (L_i, A, \theta, B)$;

else if $F_j$ type of $A \theta c$ then

$SC = SC \cup (L_i, A, \theta, c)$;

end if

end for

end for

end procedure

---

The normalization of the selection condition $F$ is required to provide a more efficient treatment of the queries. Moreover, the specification of the condition $F$ by means of visual interaction with the ESDR-Network is greatly simplified. Since the disjunctive normal form of a logical formula $F$ is specified as:

$$
F = (F_1 \land \ldots \land F_h) \lor (F_{h+1} \land \ldots \land F_i) \lor \ldots \lor (F_{k-j} \land \ldots \land F_k),
$$

each component (conjunction) of such expression can be considered as a layer, and all the layers are combined by means of the or logical operator. That is:

$$
F = \left(\left(\left(\ldots \left(\left(F_1 \land \ldots \land F_h\right) \lor \left(F_{h+1} \land \ldots \land F_i\right) \lor \ldots \lor \left(F_{k-j} \land \ldots \land F_k\right)\right)\right)\right).
$$

Once the selection condition $F$ has been normalized in such form, it is straightforward to specify each layer by applying the atomic conditions to the related atomic nodes.

For example, let us consider the selection condition $F = (F_1 \land F_2) \lor (F_3 \land F_4)$, where each atomic condition $F_i$ considers a comparison of an attribute with a constant value, e.g. $F_1 = A_2 \theta c_1$, $F_2 = A_4 \theta c_2$, $F_3 = A_3 \theta c_3$ and $F_4 = A_4 \theta c_4$. Figure 15 describes the application of such condition to a complex node $N$ with four attributes, that is, $X = \{A_1, A_2, A_3, A_4\}$. The first step requires the specification of the atomic conditions of layer 1, that is, $F_1$ and $F_2$. Figure 15a shows the application of $F_1$ to the atomic node $A_2$, by specifying the related atomic condition. The number near the condition represents the layer. Likewise, Figure 15b shows the application of the atomic condition $F_2$, while Figure 15c shows the complete application of the whole condition $F$. Note that while specifying conditions belonging to layer 2:
Figure 15: Selection.

- the atomic conditions of layer 1 are grayed, and
- the atomic nodes that do not have any atomic condition specified for the active layer are not selected (see attribute $A_2$).

It is worth noting that, in the examples of Figure 15, it is assumed that the application of an atomic condition to an atomic node causes its selection, and consequently the application of the projection operator. However, selection conditions can also be specified for an atomic node without its selection. Figure 16 shows an example:

- attribute $A_1$ is not selected since it has no selection conditions specified;
- attribute $A_2$ has a condition, but the node is not selected, since the condition is specified on layer 1 that is not active (the labels are grayed);
- attribute $A_3$ has a condition specified on layer 2, but the node is not selected: as a consequence, its condition will be evaluated during the query processing, but the resulting set of tuples will not include information related to that attribute, since it is not part of the projection subset.

Figure 16: Selection.

As a consequence, the selection of atomic nodes during the specification of selection conditions causes the implicit application of the projection operator. Since the atomic nodes can be differently selected for each layer during the specification of the selection condition, a projection subset of attributes is defined for each layer of the selection condition. That is, recalling that a selection condition composed of $n$ atomic conditions and $m$ layers is in the form:

$$F = (F_1 \land F_2 \land F_3) \lor (F_4 \land F_5 \land F_6) \lor \ldots \lor (F_{n-2} \land F_{n-1} \land F_n),$$

it corresponds to the definition of $m$ projection subsets of attributes, one for each layer. To avoid inconsistencies on the resulting set of tuples, it is required that the selection of nodes with atomic conditions specified on multiple layers (such as node $A_4$ in Figure 16) is consistent throughout all the layers for which a selection condition is specified. That is, such nodes must be selected (or not selected) in all the layers. The application of the selection operator cannot be performed if this condition is not
satisfied. More formally, considering a generic atomic node $A_i$, with atomic conditions specified over $n$ different layers (thus, the node belongs to $n$ different projection subsets), the following condition must be satisfied:

$$(A_i \in Y_j, \ j = 1, \ldots, n) \lor (A_i \notin Y_j, \ j = 1, \ldots, n).$$

The formal description of the Select operator is provided by Algorithm 9. It is worth noting that the expression of the atomic conditions of selection condition $F$ is performed by adding to the selection conditions set $SC$ a tuple representing each atomic condition.

### 4.2.5 Join

In the following paragraphs, if not otherwise specified, the term join refers to the natural join $r_1 \bowtie r_2$ between two relations $r_1(X_1)$ and $r_2(X_2)$ as defined by the relational algebra:

$$r_1 \bowtie r_2 = \{t \mid t[X_1] \in r_1 \land t[X_2] \in r_2\}.$$

**Theta-join and equi-join**

**Relational algebra definition.** The theta-join is a binary operator defined in the relational algebra as $r_1 \bowtie_F r_2$ on two relations $r_1(X_1)$ and $r_2(X_2)$, where $F$ is a propositional formula as those defined for the selection operator. The result of the operator is a set of tuples containing all the combinations of tuples in $r_1$ and $r_2$ that satisfy the condition $F$. The theta-join operator can be expressed as a combination of the selection operator and the cartesian product:

$$r_1 \bowtie_F r_2 = \sigma_F (r_1 \times r_2).$$

Whenever the propositional formula $F$ is constituted only by equivalence atomic conditions between attributes of $r_1$ and $r_2$, the operator is named **equi-join**.

**ESDR-Network definition.** Considering the ESDR-Network model, the equi-join operator corresponds to the selection of an edge $r$ that insist between two complex nodes $N$ and $M$. Since the join condition can be derived from the underlying information of the original data source\(^1\), it is not required to be specified.

The theta-join operator corresponds to the combined and appropriate application of the equi-join and selection operators.

**Algorithm 10 Equi-join.**

**Require:** $N, M \in \mathcal{N}(DS) \cap \mathcal{C}(DS); E(N, M) \in \mathcal{E}(DS)$

**Ensure:** $E(N, M) \in ESDR_{ACTIVE}$

**procedure** EQUI-JOIN($N, M$)

activate($E(N, M)$);

**end procedure**

It is worth noting that in the ESDR-Network model the basic join operator is the equi-join. Considering the definition of the model it is in fact straightforward to define the equi-join operator by analyzing the foreign key relationships mapped in the SetOfMappings.

Figure 17 shows an example of application of the equi-join operator. Given two complex nodes $N$ and $M$, each one with its related set of atomic nodes (attributes), assume that the edge $r$ between $N$ and $M$ states that node $A_2$ is a foreign key for node $N$, thus referring to the primary key $B_1$ of node $M$. The application of the equi-join operator requires the selection of the edge $r$. The join condition (i.e., FROM $N$ JOIN $M$ ON $N.A2 = M.B1$) is not specified since it can be automatically derived from the

\(^1\)As explained in [21], such information are stored in the SetOfMappings, that describes the modifications performed on involved data sources during the abstraction process.
Algorithm 11 Theta-join.

Require: \( N, M \in \mathcal{N}_C(DS); E(N, M) \in \mathcal{E}(DS); F \) expressed in disjunctive normal form on the attributes of \( N \) and \( M \)

Ensure: \( E(N, M) \in ESDR_{ACTIVE}; F \) expressed on nodes \( N, M \) and their attributes

\[
\text{procedure} \ \text{THETA-JOIN}(N, M, F) \\
\hspace{1em} \text{activate}(E(N, M)); \\
\hspace{1em} \text{join\_select}(N, M, F); \\
\text{end procedure}
\]

Algorithm 12 Join-Selection.

Require: \( N, M \in \mathcal{N}_C(DS); F \) expressed in disjunctive normal form

Ensure: \( N, M \in ESDR_{ACTIVE}; F \) expressed on nodes \( N, M \) and their attributes

\[
\text{procedure} \ \text{JOIN-SELECT}(N, M, F) \\
\hspace{1em} ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup N; \\
\hspace{1em} ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup M; \\
\hspace{1em} \text{for all selection layers } L_i \in F \text{ do} \\
\hspace{2em} \text{for all atomic conditions } F_j \in L_i \text{ do} \\
\hspace{3em} \text{if } F_j \text{ typeof } A \theta B \text{ then} \\
\hspace{4em} SC = SC \cup (L_i, A, \theta, B); \\
\hspace{4em} ESDR_{ACTIVE} = ESDR_{ACTIVE} \cup E(A, B, \langle \theta \rangle); \\
\hspace{3em} \text{else if } F_j \text{ typeof } A \theta c \text{ then} \\
\hspace{4em} SC = SC \cup (L_i, A, \theta, c); \\
\hspace{2em} \text{end if} \\
\hspace{2em} \text{end for} \\
\hspace{1em} \text{end for} \\
\text{end procedure}
\]

SetOfMappings. It is worth noting that the application of the equi-join operator should be typically coupled with the application of the projection operator, since, otherwise, no values would be returned. The situation depicted in Figure 17 assumes that all the attributes of nodes \( N \) and \( M \) should be part of the resulting set of tuples.

![Figure 17: Equi-join.](image)

The application of the theta-join operator is shown in Figure 18, assuming that the edge \( r \) has the same meaning than in Figure 17. The application of the equi-join operator is straightforward as in the previous example. However, a selection condition requires that the resulting tuples must satisfy the condition \( N.A_4 \geq M.B_4 \). As it can be noted, the selection condition is shown as usual near the atomic node \( A_4 \) but, since it involves also another atomic node (namely, \( B_4 \)), an edge labeled with the comparison operator \( \geq \) connects the two atomic nodes. It is worth noting that, in this case, not all the attributes of node \( M \) have been selected.

The formal specification of Equi-Join and Theta-Join operators is described by Algorithms 10 and 11. Algorithm 12 describes the application of a selection condition \( F \) on more than one node, as required.
by the Theta-Join operator.

**Left, Right and Full join**

**Relational algebra definition.** In the relational algebra the left join is defined as \( r_1 \bowtie_{\text{LEFT}} r_2 \) on two relations \( r_1(X_1) \) and \( r_2(X_2) \). The result of the operator is the set of tuples of relation \( r_1 \) defined on the attribute set \( X_1 \times X_2 \), combined where possible with the tuples of relation \( r_2 \), and otherwise filled with a NULL value:

\[
  r_1 \bowtie_{\text{LEFT}} r_2 = r_1 \cup (r_1 \bowtie r_2).
\]

The right join is defined as \( r_1 \bowtie_{\text{RIGHT}} r_2 \) on two relations \( r_1(X_1) \) and \( r_2(X_2) \). The result of the operator is the set of tuples of relation \( r_2 \) defined on the attribute set \( X_1 \times X_2 \), combined where possible with the tuples of relation \( r_1 \), and otherwise filled with a NULL value:

\[
  r_1 \bowtie_{\text{RIGHT}} r_2 = r_2 \cup (r_1 \bowtie r_2).
\]

The full join is defined as \( r_1 \bowtie_{\text{FULL}} r_2 \) on two relations \( r_1(X_1) \) and \( r_2(X_2) \). The result of the operator is the set of all the tuples of relations \( r_1 \) and \( r_2 \) defined on the attribute set \( X_1 \times X_2 \), combined where possible on common attribute names, and otherwise filled with a NULL value:

\[
  r_1 \bowtie_{\text{FULL}} r_2 = r_1 \cup r_2 \cup (r_1 \bowtie r_2).
\]

**ESDR-Network definition.** Considering the ESDR-Network model, the left, right and full join correspond to the selection of an edge \( r \) that insist between two complex nodes \( N \) and \( M \). The three operators are distinguished by labeling the edge with \( L \) for the left join operator, \( R \) for the right join operator and \( F \) for the full join operator. As for the equi-join operator, the join condition is derived from the SetOfMappings and it is not required to be specified.

Figure 19 shows an example of application of the three operators (left, right and full join, in Figure 19a, 19b and 19c, respectively). For the sake of simplicity, the attributes of nodes \( N \) and \( M \) are not shown. Algorithm 13 presents the formal description of the Outer-Join operator.
Algorithm 13 Outer-join.

Require: $N, M \in \mathcal{NC}(DS); E(N, M, \langle r \rangle) \in \mathcal{E}(DS); F$ expressed in disjunctive normal form

Ensure: $E(N, M, \langle r, \text{join_type}, \text{result_name} \rangle) \in \mathcal{ESDRACTIVE}; F$ expressed on nodes $N, M$ and their attributes

procedure Outer-join($N, M, F$)

\[ \text{ESDRACTIVE} = \text{ESDRACTIVE} \cup E(N, M, \langle r, \text{join_type}, \text{result_name} \rangle); \]

\[ \text{join_select}(N, M, F); \]

end procedure

4.3 Query Specification

The specification of a query over an ESDR-Network consists in two distinct phases. The first step consists in the application of the relational algebra operators defined in the previous paragraphs on the ESDR-Network. The second step consist in the execution of the query and the extraction of the results that have to be returned to the user. Figure 20 shows the different steps of the query specification process.

The example in Figure 20 describes the case in which a user wants to extract all the persons that are somehow related to an academic context. Given the ESDR-Network, in which the two concepts Professor and Researcher are specializations of the more general concept Person, the specification of the query requires to activate the nodes Professor and Researcher. As a consequence, they are added to the active layer $\text{ESDRACTIVE}$. Since the user requires the set of both professors and researchers, the Union operator is applied, by connecting the two nodes with a suitable edge. The Union operator also requires the name to be assigned to the result (in this case, Academic). Since the attributes of Professor and Researcher are not compatible, the application of the Rename operator is required, in order to make the attribute sets of the two nodes equivalent (please note that it is implicitly assumed that the domain of the attribute pairs has been considered to be compatible; for example, it can be assumed that each attribute is defined to be a string of characters). The query result will be a node labeled Academic with two attributes, namely University and Office Room.

5 Conclusions and Future Work

This report introduces the ESDR-Network conceptual data model and a related query language with the aim of providing designers an effective mean to represent an application domain and perform data tailoring operations through a direct visual interaction.

The ESDR-Network conceptual data model has been defined in order to represent integrated data sources and being able to translate back to the original data sources the queries specified on the application domain representation. The ESDR-Network conceptual data model is based on the SDR-Network data model [25], but introduces some new features (e.g., generalization edges or possibility to connect two concepts by means of more than one edge) in order to preserve all the information needed to correctly perform data tailoring.

Along with the conceptual data model, a query language has been defined in order to provide an effective way to specify in a native language intensive queries onto instantiations of the ESDR-Network conceptual data model. It has been discussed how the standard operators of the relational algebra can be defined in terms of visual interaction with the ESDR-Network conceptual data model, by providing suitable operators to express queries. As a consequence, the expressive power of the proposed query language is shown to be at least the one of the relational algebra.

Two main future work directions can be identified by considering the results obtained. In particular, a first area of improvement is the development of a framework that allows the practical use of the ESDR-Network conceptual data model. A prototype allowing the visual design of an application domain by means of the ESDR-Network conceptual data model has already been realized, while the implementation of a complete system providing the possibility to specify queries on the modeled application domains is under development.

A second direction in future work is the comparison of the expressive power of the proposed query
language with respect to other visual query languages, for example with the one proposed by Catarci and Tarantino [6] to define queries on Structure modeling Hypergraphs.

References


### Relational Model vs. SMH Data model

<table>
<thead>
<tr>
<th>Relational Model</th>
<th>SMH Data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set of domains ( D = { D_1, \ldots, D_n } )</td>
<td>Set of nodes ( N_D = { d_1, \ldots, d_n } )</td>
</tr>
<tr>
<td>Domain values ( D_i = { v_{1i}, \ldots, v_{li} } )</td>
<td>Set of nodes ( N_V = { v_1, \ldots, v_l } )</td>
</tr>
<tr>
<td>Universe of attributes ( U = { A_1, \ldots, A_m } )</td>
<td>Range ( \lambda(N_U) )</td>
</tr>
<tr>
<td>Relation names ( RN = { rn_1, \ldots, rn_k } )</td>
<td>Set of nodes ( N_{RN} = { rn_1, \ldots, rn_k } )</td>
</tr>
<tr>
<td>Domain structure function ( \text{dom} : U \rightarrow D )</td>
<td>Set of arcs ( A_{Dom} )</td>
</tr>
<tr>
<td>Relation scheme function ( \text{attr} : RN \rightarrow \mathcal{P}(U)^2 )</td>
<td>Set of hyperarcs ( \mathcal{H}_{Attr} )</td>
</tr>
<tr>
<td>Relation instance function ( \text{inst} : RN \rightarrow \mathcal{P}(t(U)) )</td>
<td>Set of hyperarcs ( \mathcal{H}_{Inst} )</td>
</tr>
<tr>
<td>Tuple ( t_i(R) )</td>
<td>Hyperedge ( h_{tuple} \in \mathcal{H}<em>{Tuple} : \text{range of } \lambda[h</em>{tuple}] = R )</td>
</tr>
<tr>
<td>Relation ( r = \langle \text{rn}, \text{attr}(	ext{rn}), \text{inst}(	ext{rn}) \rangle )</td>
<td>Relation ( r = \langle \text{rn}, h_{\text{attr}}, h_{\text{inst}} \rangle )</td>
</tr>
</tbody>
</table>

Table 1: Mappings between the relational data model and the relational SMH model.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Action</th>
<th>Specialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in h_{Sch}$, with $h_{\text{Attr}} = \langle n, h_{Sch} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext}}$, with $h_{\text{Inst}} = \langle n, h_{\text{Ext}} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall h_{\text{tuple}} \in db_{ON} h_{Eq} := h_{Eq} \cup h'_{\text{ Tuple}}$</td>
</tr>
<tr>
<td>Union</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in h_{Sch1}$, with $h_{\text{Attr}} = \langle n, h_{Sch1} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext1}} \cup h_{\text{Ext2}}$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall h_{\text{tuple}} \in db_{ON} h_{Eq} := h_{Eq} \cup h'<em>{\text{ Tuple}}$, such that $h'</em>{\text{ Tuple}} \approx h_{\text{ Tuple}}$</td>
</tr>
<tr>
<td>Difference</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in h_{Sch1}$, with $h_{\text{Attr}} = \langle n, h_{Sch1} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext1}} - h_{\text{Ext2}}$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall h_{\text{tuple}} \in db_{ON} h_{Eq} := h_{Eq} \cup h'<em>{\text{ Tuple}}$, such that $h'</em>{\text{ Tuple}} \approx h_{\text{ Tuple}}$</td>
</tr>
<tr>
<td>Projection</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in X$, where $X \subseteq h_{Sch1}$, with $h_{\text{Attr}} = \langle n, h_{Sch1} \rangle$, and $X$ is the set of nodes corresponding to attributes in $A$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext1}}$, with $h_{\text{Inst1}} = \langle n, h_{\text{Ext1}} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall h_{\text{tuple}} \in db_{ON} h_{Eq} := h_{Eq} \cup h'<em>{\text{ Tuple}}$, such that $h'</em>{\text{ Tuple}} \subseteq h_{\text{ Tuple}}$</td>
</tr>
<tr>
<td>Selection</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in h_{Sch1}$, with $h_{\text{Attr1}} = \langle n, h_{Sch1} \rangle$ and $X$ is the set of nodes corresponding to attributes in $A$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext1}}$, with $h_{\text{Inst1}} = \langle n, h_{\text{Ext1}} \rangle$ which verifies the selection condition $F$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall h_{\text{tuple}} \in db_{ON} h_{Eq} := h_{Eq} \cup h'<em>{\text{ Tuple}}$, such that $h'</em>{\text{ Tuple}} \approx h_{\text{ Tuple}}$</td>
</tr>
<tr>
<td>Product</td>
<td>Structure def.</td>
<td>$\text{on}(db,n) \forall n \in h_{Sch1} \cup h_{Sch2}$, with $h_{\text{Attr1}} = \langle n, h_{Sch1} \rangle$ and $h_{\text{Attr2}} = \langle n, h_{Sch2} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Extension def.</td>
<td>$\text{on}(db,h_{\text{tuple}}) \forall h_{\text{tuple}} \in h_{\text{Ext1}} \cup h_{\text{Ext2}}$</td>
</tr>
<tr>
<td></td>
<td>Extension comp.</td>
<td>$\forall (h_{\text{Tuple1}}, h_{\text{Tuple2}}) \in db_{ON}$ with $h_{\text{Tuple1}} \in h_{\text{Ext1}}$ and $h_{\text{Tuple2}} \in h_{\text{Ext2}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$h_{Eq} := h_{Eq} \cup h'<em>{\text{ Tuple}}$, such that $h'</em>{\text{ Tuple}} = h_{\text{Tuple1}} \times h_{\text{Tuple2}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$h'<em>{\text{Tuple}}$ and $h</em>{Eq}$ satisfy the constraints</td>
</tr>
</tbody>
</table>

Table 2: Mappings between the relational algebra operators and the RSMH algebra.