WPI-CS-TR-97-8 September 1997

CVS: The Complex Substitution Algorithm for View Synchronization

by

Anisora Nica
Amy J. Lee
Elke A. Rundensteinter

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

CVS: The Complex Substitution Algorithm for View Synchronization*

Anisoara Nica[†], Amy J. Lee[†], and Elke A. Rundensteiner[‡]

(†) Department of EECS
University of Michigan, Ann Arbor
Ann Arbor, MI 48109-2122
amylee,anica@eecs.umich.edu

(313) 764-1571

(‡)Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
rundenst@cs.wpi.edu

(508) 831-5815

February 24, 1998

Abstract

Maintenance of materialized views in large-scale information spaces such as the WWW has importance for many applications. However, such large-scale environments typically contain autonomous information sources (ISs) that dynamically modify their content, interfaces as well as their query services. Unfortunately, current view technology supports only static views in the sense that views become undefined and hence obsolete as soon as the underlying ISs undergo capability changes. We propose to address this view evolution problem by a novel solution approach that allows view definitions to be dynamically evolved when they become undefined. In particular, we have enhanced the (SQL-based) view definition language with evolution parameters to allow the view definer to state her view evolution preferences, indicating for example which components of the view are completely dispensable, replaceable by alternate components, etc. Furthermore, in our view synchronization architecture, a description of the content, capabilities as well as interrelationships of all ISs is maintained in a meta-knowledge base (MKB) in order to provide this knowledge to the view synchronization process. Given a view modeled by the extended view definition language and the MKB, we present in this paper a formal definition of what corresponds to legal rewritings of a view affected by capability changes. Based on this formal foundation, we then propose a three-step strategy for the view synchronization process. Given a capability change, we propose algorithms for (1) evolving the meta knowledge base (MKB) itself, (2) detecting views possibly affected by this MKB evolution, and (3) evolving the affected view definitions guided by constraints imposed by the view evolution preferences as well as by the knowledge captured in the MKB. The proposed strategy can be shown to find a new valid definition of a view in many cases where current view technology would have simply disabled the view.

Keywords: Self-adapting views, view synchronization and preservation, data warehouse, large-space information space, information descriptions, evolving information sources.

*This work was supported in part by the NSF RIA grant #IRI-9309076 and NSF NYI grant #IRI 94-57609. We would also like to thank our industrial sponsors, in particular IBM for the IBM Partnership Award and our collaborators at IBM Toronto for their support.

1 Introduction

Advanced applications such as web-based information services, data warehousing, digital libraries, and data mining typically operate in an information space populated with on a large number of dynamic information sources (ISs) [Wid95]. The ISs in such environments are usually distributed, have distinct schemas, support different query languages, update not only their content but also their capabilities[1], and even join or leave the environment frequently. In order to provide easy access to information in such environments, relevant data is often retrieved from several sources, integrated as necessary, and then materialized at the user site. For instance, many WWW users may be interested in all information related to travel including fares, special bargains and flight availabilities of different airlines. While such information could principally be retrieved by each of the interested users by querying over many ISs and integrating the results into a meaningful answer, it is much preferable if one *travel consolidator service* were to collect such travel-related information from different sources on the WWW and to organize such information into a materialized view. Besides providing simplified information access to users who may not have the time nor skill to identify and retrieve relevant information from all sources on the WWW, such a materialized view may also offer more consistent availability shielding users from the fact that some of the underlying ISs may temporarily become disconnected as well as better query performance.

Materialized views in such evolving environments introduce new challenges to the database community [Wid95]. In our prior work [RLN97], we have identified view evolution as a critical new problem faced by these applications. The problem is that current view technology is insufficient for supporting *flexible* view definitions. That is, under current view technology, views are *static*, meaning views are assumed to be specified on top of a fixed environment and once the external ISs change their capabilities, the views defined upon them become undefined. In our prior work, we have proposed a novel approach to solve this view inelasticity problem [LNR97a]. Namely, we have designed a framework for view maintenance in these evolving environments which supports to "preserve as much as possible" of the view instead of completely disabling it with each IS change. While the evolution of views is assumed to be triggered by capability changes of ISs in our work, previous work (e.g., by Gupta et al. [GMR95] and Mohania et al. [MD96]) typically assumed that view redefinition at the schema level was explicitly requested by the view developer at the view site. Furthermore, previous work Gupta et al. [GMR95], Mohania et al. [MD96], Huyn [Huy96], etc., has focused on maintenance of the data itself (i.e., content changes) that is associated with the modified schema and not with modifications of the view definitions themselves as done in this paper.

One key component of our solution is the extended view definition language (essentially, SQL extended with view evolution preferences) that allows the view definer to control the view evolution triggered by environment changes by indicating the criticality and dispensability of the different components of the view definition. For example, a view definer could indicate that the attribute **Name** is indispensable to the view, whereas the attribute **Address** is desirable yet can be omitted from the original view definition, if keeping it becomes impossible, without jeopardizing the utility of the view.

A second key component of our solution is a language for capturing a description of the content, capabilities as well as interrelationships of all ISs in the system. In order to keep our approach general we only consider basic types of constraints in our model that are likely to be applicable to many types of information sources. For

[1] Capabilities here refer to information such as their schema, their query interface, as well as other services offered by the information source.

this reason, constraints such as keys and functional dependencies are assumed not to be available by most ISs and hence are not supported in our model. This IS description is maintained in a meta-knowledge base (MKB) available to the view synchronizer during the view evolution process.

Given a view modeled by the extended view definition language and given a MKB, we now present in this paper a formal foundation for what corresponds to legal rewritings of the view affected by capability changes. This includes properties characterizing that all MKB constraints must be obeyed, as well as that maximal preservation of the view including preferences given by the view definer must be achieved by the view redefinition process.

Based on this formal foundation, we then propose a three-step strategy for the view synchronization process. Given a capability change, we first propose algorithms for evolving the meta knowledge base (MKB) itself. Second, we introduce an algorithm for detecting views affected by this MKB evolution. Lastly, we present the view synchronization algorithm that evolves the affected view definitions guided by constraints imposed by the view evolution preferences as well as by the knowledge captured in the MKB. Our view synchronization algorithm attempts to find valid replacements for affected (deleted) components of the existing view definitions based on the semantic constraints captured in the MKB. These replacements thus correspond to possibly complex pieces of information from several ISs [LNR97b]. The proposed strategy can be shown to find a new valid definition of a view in many cases where current view technology would have simply disabled the view.

While we have introduced the view evolution problem as well as the overall solution framework in [RLN97], new contributions of this paper now include the following. First, since our work discusses applications operating in a dynamic environment, issues associated with meta knowledge base management are important. To address this, we now present our solution of evolving the meta knowledge base itself under the basic set of capability changes. Second, we present a formal foundation for when view rewriting is considered to be legal within the context of our problem. For this, rather than just determining simple view rewriting, we are interested in providing possibly complex view rewrites through multiple join constraints given in MKB. Third, to demonstrate our solution approach, we present algorithms for view synchronization. In particular, we focus on handling the most difficult capability change operator, namely, the delete-relation operator, in depth in this paper.

The remainder of the paper is structured as follows. In Section 2, we present our solution approach, and introduce a web-based travel agency example to serve as running example throughout the paper. In Section 3, we present the knowledge representation model that is used to describe the capabilities of each IS as well as the interrelationships between ISs. The extended view definition language designed to address evolution preferences is presented in Section 4. The MKB evolution process is presented in Section 5. Sections 6 describes the formal basis for correct view synchronization, while Section 7 introduces our proposed algorithm for synchronizing views based on this formal model. Section 8 lists work in the literature that is most closely related to ours, and Section 9 presents our conclusions.

2 Background

In this section, we first give a brief overview of the evolvable view environment (EVE) framework that we have designed to tackle the view evolution problem (Figure 1) [RLN97]. Then, we describe a web-based travel consolidator as our running example used throughout the remainder of this paper.

2.1 Framework for View Maintenance in Evolving Environments

Our environment can be divided into two spaces, i.e., the view site and the information space. The information space is populated by a large number of heterogeneous and autonomous ISs. These ISs join, leave, or change their capabilities dynamically, thus it is of particular concern with how to maintain views in such an evolving environment. An IS is "integrated" into our proposed framework via a mediator (i.e., the information source interface (ISI)) that serves as a bridge between the information space and the view site. Any IS that supports a query interface can participate in our framework.

When an IS joins our environment, it advertises itself and registers the information regarding its capabilities and data content to the meta knowledge base (MKB). For this purpose, we have designed a model capable of describing the content and capabilities of heterogeneous and widely diverse ISs [NR97]. Our model captures the capability of each IS and the relationships between ISs by partial and/or complete containment constraints, join constraints, and attribute transformation constraints (see Section 3). The IS descriptions collected in the MKB are critical for identifying alternate view definitions when evolving a view definition.

Views and ISs within the framework are decoupled in the sense that each IS may have its own data definition and query languages. The knowledge about the views, such as the view definitions and the location of the views, is stored in the view knowledge base (VKB). One novel aspect of our solution framework, not previously addressed in any other work, is that each view in our environment is described by a view specification that incorporates evolution characteristics of views in terms of their dependence on (possibly evolving) ISs as well as the criticality of view components for the existence of the view (Section 4). These specifications, expressed as *view evolution parameters*, allow the user to specify criteria based on which the view will be transparently evolved by the system under capability changes of ISs. In this work, we concern ourselves with capability changes (i.e., schema evolution operations) typically found in literature, such as adding/deleting an attribute, changing name of an attribute, adding/deleting a relation, and changing name of a relation.

View synchronization is concerned with synchronizing the view definition with the modified IS descriptions so that the view remains both valid and consistent with the view evolution parameters and with all IS constraints kept in the MKB. Upon receiving a capability change notification at an IS, the *view synchronizer* tool (see Figure 1) identifies which views are affected by the capability change based on knowledge captured in the VKB (see algorithms in Sections 5 and 6). Then, based on the view change semantics expressed by our extended view definition language E-SQL, the *view synchronizer* explores alternate techniques for query rewriting with the goal of meeting all view preservation constraints in the view definition (VD), extracting appropriate information from other ISs as replacement of the modified capabilities using the MKB, and then calculating the new view definition (VD') (see Section 6).

2.2 Running Example

Below, we describe a web-based travel consolidator as running example used throughout the remainder of this paper.

Example 1 *Consider a large travel agency which has a headquarter in Detroit, USA, and many branches all over the world. It helps its customers to arrange flights, car rentals, hotel reservations, tours, and purchasing insurances. Therefore, the travel agency needs to access many disparate information sources, including domestic as well as international sites. Since the connections to external information sites, such as the overseas*

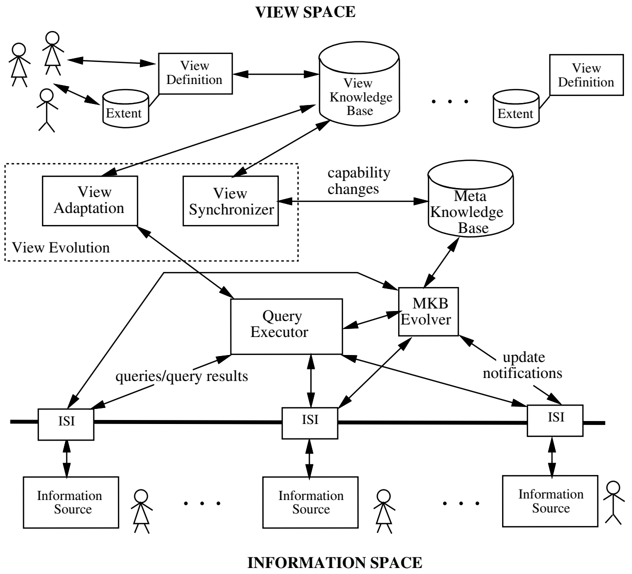

Figure 1: The EVE Framework: View Evolution in a Changing Environment.

IS 1: Customer Information **Content Description: Customer(Name, Address, Phone, Age)**
IS 2: Tour Information **Content Description: Tour(TourID, TourName, Type, NoDays)**
IS 3: Tour Participant Information **Content Description: Participant(Participant, TourID, StartingDate,Location)**
IS 4: Flight Reservation Information **Content Description: FlightRes(Passenger, Airline, FlightNo, Source, Destination, Date)**
IS 5: Insurance Information **Content Description: Accident−Ins(Holder, Type, Amount, Birthday)**
IS 6: Hotel Information **Content Description: Hotels(City, Address, PhoneNumber)**
IS 7: Car Rental Company Information **Content Description: RentACar(Company, City, PhoneNumber, Location)**

Figure 2: Information Source Content Descriptions.

branches, are very expensive and have low availability, the travel agency materializes the query results (views) at its headquarter or other US branches (at the view site). The information space is summarized in Figure 2.

Assume the headquarter maintains complete information of the customers ***Customer(Name, Address, Phone, Age)****, the complete set of* ***Tour*** *information* ***Tour(TourID, TourName, Type, NoDays)*** *- where* ***Type*** $= \{luxurious, economy, super\text{-}valued\}$*, and the information* ***Participant(Participant, TourID, StartingDate)*** *stating which customer joins which tour starting on what day. We further assume the local branches keep partial information of its local customers, the tours offered locally, and the participation information of its local customers. The flight reservation information* ***FlightRes(Passenger, Airline, FlightNo, Source, Destination, Date)*** *is managed by each individual airline company. Insurance information* ***Accident−Ins(Holder, Type, Amount, Birthday)*** *is kept with each individual insurance company.*

3 Information Sources Description Language

While an individual IS can be based on any data model, we assume that the information source interface (ISI) agent of an IS describes the schema exported by the IS as a set of relations $IS.R_1$, $IS.R_2, \ldots, IS.R_n$, each with two or more attributes. Note that a relation name is unique within each IS, but may not be unique across ISs. We assume the combination of (IS name, relation name) is unique in the MKB. The descriptions of an IS contains information of its data structure and content, its query capabilities, and its relationships with exported relations from other ISs. We assume in this paper that all ISs support at least SQL queries composed of a conjunction of primitive clauses in the WHERE clause, and we do not discuss how query capabilities are described in our system further. (For a description of an IS description model that including query capabilities see [NR97].)

3.1 Type Integrity Constraints

Each attribute A_i is given a name and a data type to specify its domain of values. This information is specified by using a type integrity constraint with the format $Type_i(A_i)$. A *type constraint* for a relation $R(A_1, \ldots, A_n)$ is specified as:

$$R(A_1, \ldots, A_n) \subseteq Type_1(A_1), \ldots, Type_n(A_n) \tag{1}$$

which says that the attribute A_i is of type $Type_i(A_i)$. For simplicity, we assume that the attribute types are primitive. If two attributes are exported with the same name, they are assumed to have the same data type (which must be reflected by the type constraints for their relations)[2].

3.2 Order Integrity Constraints

The *order constraints* specify data constraints that are satisfied by any tuple of a relation at any time. For a relation $R(A_1, \ldots, A_n)$, an order constraint is defined as follows:

$$R(A_1, \ldots, A_n) \subseteq \mathcal{C}(A_{i_1}, \ldots, A_{i_k}) \tag{2}$$

where $(A_{i_1}, \ldots, A_{i_k})$ is a subset of the attributes of R and $\mathcal{C}(A_{i_1}, \ldots, A_{i_k})$ is a conjunction of primitive clauses defined over the attributes. A primitive clause has one of the following forms: (*<attribute-name>* θ *<attribute-name>*) or (*<attribute-name>* θ *<value>*) with $\theta \in \{<, \leq, =, \geq, >\}$. Expression (2) specifies that for any state of the database R, and for any tuple $t \in R$, $\mathcal{C}(t[A_{i_1}], \ldots, t[A_{i_k}])$[3] is satisfied.

Example 2 *An insurance relation **Expensive-Insurance**, containing all expensive accidental insurances that cover more than* \$1,000,000*, can be expressed by the following order constraint:*

Expensive-Insurance(Holder, Type, Amount, Birthday) $\subseteq$ (***Amount*** $> 1,000,000$).

[2] In the future work we plan to allow complex types and utilize a hierarchy of types. We anticipate that most of the proposed solution approach will continue to apply to these extended types.

[3] The expression $t[A]$ refers to the value of the attribute A in the tuple t.

3.3 Join Constraints

A *join constraint* is used to specify a meaningful way to combine information from two ISs. The join condition is a conjunction of primitive clauses C_i (not necessarily equijoin clauses). Formally, a join constraint is of the form:

$$\mathcal{JC}_{R_1,R_2} = (C_1 \cdots \; AND \; \cdots C_l) \tag{3}$$

where $C_1, \ldots, C_l$ are primitive clauses over the the attributes of R_1 and R_2. Expression (3) gives a default, legal join condition used to join R_1 and R_2, that is the joined relation $J = R_1 \bowtie_{(C_1 \cdots \; AND \; \cdots C_l)} R_2$ is a meaningful way of combining the two relations.

Example 3 *The following join constraint defined for the relations* ***Customer(Name, Address, Phone, Age)*** *and* ***Person(SSN, Name, Address)*** *states that the two relations can be joined on the attributes* ***Name*** *and* ***Address****:*

$$\mathcal{JC}_{\boldsymbol{Person,Customer}} = (\boldsymbol{Person.Name} = \boldsymbol{Customer.Name} \; AND \; \boldsymbol{Person.Address} = \boldsymbol{Customer.Address})$$

For our running Example 1, the join constraints are given in Figure 3.

$\mathcal{JC}$ Name	Relations	Join Constraint
JC1	**Customer, FlightRes**	**Customer.Name = FlightRes.Passenger**
JC2	**Customer, Accident−Ins**	**Customer.Name = Accident−Ins.Holder**
JC3	**Customer, Participant**	**Customer.Name = Participant.Participant**
JC6	**FlightRes, Accident−Ins**	**FlightRes.Passenger = Accident−Ins.Holder**
JC4	**Participant, Tour**	**Participant.TourID = Tour.TourID**
JC5	**Hotels, RentACar**	**Hotels.Address = RentACar.Location**

Figure 3: Join Constraints for Example 1

3.4 Partial and Complete Information Constraints

The *partial and complete* ($\mathcal{PC}$) information constraints make it possible to describe that a fragment of a relation is part of or equal to a fragment of another relation for all extents of the two relations. The $\mathcal{PC}$ information constraints are used to decide if an evolved view is equivalent, subset of or superset of the initial view. For two relations R_1 and R_2, the $\mathcal{PC}$ information constraint is given by:

$$\mathcal{PC}_{R_1,R_2} = \Big(\pi_{R_1.A_{i_1},\ldots,R_1.A_{i_k}}(\sigma_{\mathcal{C}(A_{j_1},\ldots,A_{j_l})}R_1) \; \theta \; \pi_{R_2.A_{n_1},\ldots,R_2.A_{n_k}}(\sigma_{\mathcal{C}(A_{m_1},\ldots,A_{m_t})}R_2)\Big) \tag{4}$$

where θ is $\{\supseteq, \equiv\}$ for the partial or complete information constraint, respectively; $R_1.A_{i_1}$, …, $R_1.A_{i_k}$, $R_1.A_{j_1}$, …, $R_1.A_{j_l}$ are attributes of R_1; and $R_2.A_{n_1}$, …, $R_2.A_{n_k}$, $R_2.A_{m_1}$, …, $R_2.A_{m_t}$ are attributes of R_2. The sets $R_1.A_{i_1}, \ldots, R_1.A_{i_k}$ and $R_2.A_{n_1}, \ldots, R_2.A_{n_k}$ are such that any attribute $R_1.A_{i_s}$, for $s = 1, \ldots, k$, has the same type as $R_2.A_{i_s}$.

Example 4 *The Equation (5) states that the relations* ***Person*** *and* ***Customer*** *have the same data for the attributes* ***Name*** *and* ***Address*** *for customers age* 1 *or older. (The relation* ***Person*** *contains* ***SSN*** *only for persons who are older than* 1 *year.)*

$$\mathcal{PC}_{\boldsymbol{Person, Customer}} = \left(\pi_{\boldsymbol{Name,\ Address}}(\boldsymbol{Person}) = \pi_{\boldsymbol{Name,\ Address}}(\sigma_{Age>1}\boldsymbol{Customer})\right)$$

Using the $\mathcal{PC}$ information constraints and type constraints we can, for example, define that two relations are equivalent: (1) they have attributes of the same types (expressed by type constraints); and (2) their extents are the same (expressed by PC constraints).

3.5 Attribute Function-Of Constraints

The *attribute function-of constraint* relates two attributes by defining a function to transform one of them into another. This constraint is specified by:

$$\mathcal{F}_{R_1.A, R_2.B} = (\ R_1.A = f(R_2.B)\ ) \tag{5}$$

where f is a function [4]. The function-of constraint $\mathcal{F}_{R_1.A,R_2.B}$ specifies that if there exists a meaningful way of combining the two relations R_1 and R_2 (e.g., using join constraints) then for any tuple t in the joined relation J, we have $t[R_1.A] = f(t[R_2.B])$.

Example 5 *Let* ***Age****, an attribute for the relation* ***Customer*** *(C), be a customer's age, and* ***Birthday****, an attribute for the relation* ***Insurancei*** *(I), be an insurance policy holder's birthday. Once we identify* $c \in$ ***Customer*** *is the same as* $i \in$ ***Insurance****, we have:*

$$\mathcal{F}_{\boldsymbol{C.Age, I.Birthday}} = (\boldsymbol{C.Age} = (today - \boldsymbol{I.Birthday})/\ 365)$$

4 Extending SQL for Flexible View Evolution

We have enhanced the conventional view definition language (SQL) with evolution parameters to allow the view definer to state her view evolution preferences. We call this enhanced language the *evolvable-SQL* view definition language (or E-SQL for short). In this section, we briefly describe this extended view definition language. Evolution preferences are expressed as *view evolution parameters*, which allow the user to specify criteria based on which the view will be transparently evolved by the system under capability changes at the ISs. We assume SELECT-FROM-WHERE views defined as in Equation (6), with a conjunction of primitive clauses in the WHERE clause. Each component of the view definition (i.e., attribute, relation or condition) has attached two evolution parameters: the *dispensability parameter* specifies if the component could be dropped (value true) or must be present in any evolved definition (value false); the *replaceability parameter* specifies if the component could be replaced in the process of view evolution (value true) or must be left unchanged as defined in the initial view (value false).

We summarize the evolution parameters in Figure 4. Each row represents one type of evolution parameter in our view language. Figure 4 has four columns: column one gives the name, column two the symbol, column

[4] Note that the inverse of f is not required to exist, and hence if an inverse is available it must be explicitly listed as $R_2.B = f^{-1}(R_1.A)$.

three the semantics, and column four the default value. When the parameter setting is omitted from the view definition, then the default value is assumed. This means that a conventional SQL query (without evolution preferences) has well-defined change semantics in our system. That is, a view definition with all the evolution parameter values having the default values means the relations in the FROM clause cannot be substituted with other relations, and the attributes and conditions in the SELECT and WHERE clauses are not allowed to be dropped. This default setting of view evolution corresponds to the most rigid information preservation requirement, i.e., anything the user specified in the view definition must be preserved.

Parameter	Symbol	Semantics	Default
Attribute-dispensability	π	*true:* the attribute is dispensable *false:* the attribute is indispensable	false
Attribute-replaceability	ε	*true:* the attribute is replaceable *false:* the attribute is nonreplaceable	false
Condition-dispensability	σ	*true:* the condition is dispensable *false:* the condition is indispensable	false
Condition-replaceability	β	*true:* the condition is replaceable *false:* the condition is nonreplaceable	false
Relation-dispensability	α	*true:* the relation is dispensable *false:* the relation is indispensable	false
Relation-replaceability	θ	*true:* the relation is replaceable *false:* the relation is nonreplaceable	false
View-extent	Δ	don't care: the new extent can be anything $\equiv$: the new extent is equal to the old extent $\supseteq$: the new extent is a superset of the old extent $\subseteq$: the new extent is a subset of the old extent	$\equiv$ $\equiv$

Figure 4: View Evolution Parameters.

The format of the E-SQL view definition language is as follows:

$$\begin{array}{ll}
\mathsf{CREATE\ \ VIEW} & V\ \ (B_1, \ldots, B_m)\ \ (\Delta\ =\ \Delta_v)\ \mathsf{AS} \\
\mathsf{SELECT} & R_1.A_{s_{1,1}}(\pi = \pi_{s_{1,1}}, \epsilon = \epsilon_{s_{1,1}}), \ldots, R_1.A_{s_{1,i_1}}(\pi = \pi_{s_{1,i_1}}, \epsilon = \epsilon_{s_{1,i_1}}), \ldots, \\
 & R_n.A_{s_{n,1}}(\pi = \pi_{s_{n,1}}, \epsilon = \epsilon_{s_{n,1}}), \ldots, R_n.A_{s_{n,i_n}}(\pi = \pi_{s_{n,i_n}}, \epsilon = \epsilon_{s_{n,i_n}}) \\
\mathsf{FROM} & R_1(\alpha = \alpha_1, \theta = \theta_1), \ldots, R_n(\alpha = \alpha_n, \theta = \theta_n) \\
\mathsf{WHERE} & C_1(\sigma = \sigma_1, \beta = \beta_1), \ldots, C_k(\sigma = \sigma_k, \beta = \beta_k)
\end{array} \tag{6}$$

where the set $\{A_{s_{j,1}}, \ldots, A_{s_{j,i_j}}\}$ is a subset of the attributes of relation R_j.

Next, we use one example to demonstrate the integrated usage of and interactions among these evolution parameters, while a more extensive justification for the design of this language plus many more examples can be found in [LNR97b].

Example 6 *In our Example 1, let's assume that the travel agency has a promotion for the customers who travel to Asia. Therefore, the travel agency needs to find the customers' names, addresses, and phone numbers. The travel agency is either going to send promotion letters to these customers or call them by phone. The query for getting the necessary information can be specified as follows:*

$$\begin{array}{ll}
\textit{CREATE\ \ VIEW} & \textit{Asia-Customer AS} \\
\textit{SELECT} & \textit{Name, Address, PhoneNo} \\
\textit{FROM} & \textit{Customer C, FlightRes F} \\
\textit{WHERE} & \textit{(C.Name = F.Passenger) AND (F.Destination = 'Asia')}
\end{array} \tag{7}$$

Equation (7) is a static SQL query, incapable of evolving, which is inevitable in a dynamic environment. Next, we reformulate Equation (7) with view evolution parameter so that it can be evolved when the environment changes.

Assume the travel agency is willing to accept the query results with the customer's names and addresses only. That is, the company is okay to put off the phone marketing strategy, if the customer's phone number attribute ***PhoneNo*** *is deleted from the relation* ***Customer*** *for some reason and a suitable substitute cannot be found. The user can state this preference clearly in the SELECT clause (Equation (7)) by using the attribute dispensability parameter* π.

$$\mathsf{SELECT} \qquad \boldsymbol{Name}(\pi = false), \boldsymbol{Address}(\pi = false), \boldsymbol{PhoneNo}(\pi = true)$$

In addition to instructing our system that the customer name and address have to be kept in the view interface, the user may want to guide the system as to whether it is acceptable for an attribute to be obtained from other sources besides the original relation. For example, if the user only accepts the customer name and address to come from the relation ***Customer****, but agrees to have the phone number come from other source(s), then the user can augment the SELECT clause (Equation (7)) with the attribute replaceable parameter.*

$$\mathsf{SELECT} \qquad \boldsymbol{Name}(\pi = false, \varepsilon = false), \boldsymbol{Address}(\varepsilon = false), \boldsymbol{PhoneNo}\ (\varepsilon = true)$$

Further, let's assume the person who defines the view ***Asia-Customer*** *is willing to accept a view without the second (local) condition specified, as long as the equijoin condition is kept*[5]*. As a consequence (if the local condition is dropped), the promotion invitation letters are sent to all customers traveling by air. The user can explicitly specify her preference by adding the condition-dispensable parameter to the conditions in the WHERE clause of Equation (7).*

$$\mathsf{WHERE} \qquad (\boldsymbol{C.Name} = \boldsymbol{F.Passenger})\ (\sigma = false) \quad \boldsymbol{AND} \quad (\boldsymbol{F.Destination} = \boldsymbol{Asia})\ (\sigma = true)$$

If the user requires the redefined view extent to be either equivalent to or larger than the original view extent, the user sets the view-extent parameter to $\supseteq$*. This means any substitution of a relation, condition, or attribute should make the view extent equivalent to or larger than the original view extent for the view evolution process to be valid. That is, if originally the view* ***Asia-Customer*** *returns the customers who travel to Japan, Korea, or Hong Kong, then the view is still valid if in addition to these customers it also returns the customers who travel to Thailand and Malaysia.*

$$\mathsf{CREATE\ \ VIEW} \quad \boldsymbol{Asia\text{-}Customer}\ (\Delta\ = \supseteq)\ \mathsf{AS}$$

Uniting all proposed view evolution parameters with Equation (7), we get Equation (8). For the view components that have their view evolution parameter values omitted, the default value is assumed as indicated in Figure 4. To name a few, **Name** *and* **Address** *attributes in the select clause are indispensable, and the relation* **FlightRes** *is indispensable and nonreplaceable.*

[5]Note that in general dropping a local condition is more acceptable than dropping a join condition, since dropping a join condition may change the view definition dramatically. For example, replacing a join condition that returns some subset of tuples by a Cartesian product which then would return all pairwise combinations of tuples from both relations as view result.

CREATE VIEW	***Asia-Customer*** ($\Delta = \supseteq$) AS
SELECT	***Name*** ($\varepsilon = false$), ***Address*** ($\varepsilon = false$), ***PhoneNo*** (π = ***true***, $\varepsilon = true$)
FROM	***Customer C*** ($\theta = true$), ***FlightRes F***
WHERE	***(C.Name = F.Passenger)*** ($\sigma = false$) ***AND (F.Destination = 'Asia')*** ($\sigma = true$)

(8)

5 Meta Knowledge Base Evolution Under Capability Changes

First we describe what constitutes a legal MKB evolution, and then we present the MKB evolution process for all constraint types.

5.1 Foundation For Valid MKB Evolution

Let MKB be the current state of the meta knowledge base containing IS description knowledge for a given information space. Let *ch* be a capability change of one of the ISs. Let *Detect-Affected-MKB(MKB,ch)* be a boolean function that detects if the MKB is affected by the change *ch*. When *Detect-Affected-MKB(MKB,ch)* is true, the MKB is affected; and false, otherwise. When the returned value is true, our system finds and removes, as appropriate, the affected meta knowledge. Definition 1 defines the notion of a valid evolution of the MKB under a capability change ch.

Definition 1 *MKB' is a* valid evolution *of the meta knowledge base (MKB) under a capability change ch if the following properties hold:*

P1. The new meta knowledge base MKB' is no longer affected by the change ch. That is, Detect-Affected-MKB(MKB',ch) *= false.*

P2. All condition-dispensability and condition-replaceability parameters in the join constraints of the MKB are satisfied by the MKB'. For example, the MKB' must have all indispensable primitive clauses in the join constraints.

P3. The MKB' only contains information supported in the MKB, i.e., either the knowledge in the MKB' also existed (explicitly or implicitly) in the MKB or it was specified to be added to the MKB by ch*.*

P4. The MKB' is maximal with respect to the MKB, in the sense that any of the implicit or explicit meta knowledge is still accessible in the new state MKB' unless it is in conflict with ch. For example, if we have $\pi_{A,B}(R) \supseteq \pi_{A,B}(S)$ and $\pi_{A,B}(S) = \pi_{A,B}(T)$, which are the partial/complete information constraints between relation pairs (R, S) and (S, T) in the MKB – the partial/complete information constraint $\pi_{A,B}(R) \supseteq \pi_{A,B}(T)$ between R and T is an implicit meta knowledge information constraint between (R, T). When the relation S is removed by the user, the new state of the meta knowledge base MKB' should still have the meta knowledge (possibly becoming explicit) between R and T, although the partial/complete information constraints between (R, S) and (S, T) are dropped.

5.2 MKB Evolution Process

Figure 5 depicts whether a particular type of meta knowledge in the MKB may or may not be affected by a capability change at an IS. Each row represents one type of (out of six) capability change under consideration (Section 2), and each column represents the types of constraints supported by our meta knowledge model (Section 3). We make the following observations:

- Adding new attributes (the second row) or relations (the fifth row) to an IS does not affect the existing meta knowledge in the MKB, though it is likely that subsequently new meta knowledge concerning new capabilities and even the rest of the information space may be added by the information provider. We assume that our system does not attempt to further optimize existing views, when new knowledge becomes available in MKB.
- Our system keeps a name-mapping table in the MKB along with other meta knowledge. For this reason, none of the meta knowledge is affected by the change-attribute-name (the third row) and the change-relation-name (the sixth row) capability changes. Even if a name changes more than once, our system keeps track of this information in the same entry of the name mapping table.
- Constraints in our MKB are only affected by two types of capability changes, namely by deleting an attribute (the first row) or deleting a relation (the fourth row).

Based on the above assumptions, we discuss the meta knowledge evolution process under delete-attribute and delete-relation changes only in the rest of this section.

capability \ constraint change	type integrity constraint	order integrity constraint	attribute function-of constraint	join constraint	partial\complete info constraint
del-attr	maybe affected	maybe affected	maybe affected	maybe affected	maybe affected
add-attr	unaffected	unaffected	unaffected	unaffected	unaffected
chg-attr-name	unaffected	unaffected	unaffected	unaffected	unaffected
del-rel	maybe affected	maybe affected	maybe affected	maybe affected	maybe affected
add-rel	unaffected	unaffected	unaffected	unaffected	unaffected
chg-rel-name	unaffected	unaffected	unaffected	unaffected	unaffected

Figure 5: MKB Evolution under Capability Changes.

5.2.1 Integrity Constraint Evolution

As mentioned earlier, an IS describes its content by using two kinds of integrity constraints, namely the *type integrity constraints* and the *order integrity constraints*. When an attribute *attr* of a relation R is dropped by the user, then our algorithm will drop the type constraint of *attr* beside removing *attr* from the relation R. Furthermore, if an order constraint exists for R, then it will find the affected primitive clauses that reference to *attr*, compute the transitive closure of the affected clauses, and evolve the order constraint of R by adding the derived primitive clauses (if any) to the order constraint and by removing the affected primitive clauses at the end.

Example 7 *Assume the order constraint of R is: R(A,B,C) $\subseteq$ (A > B) AND (B > C). When the attribute B is dropped, our system first drops B and type$_i$(B) from the MKB and finds both clauses of the order constraint are affected. After computing the transitive closure of the affected clauses, our system derives an order constraint (A > C) that was implicitly stated in the original order constraint. Since all the (explicit and implicit) primitive clauses of the order constraint are satisfied by any extent of R at any time, the implicit order constraint (A > C) is still satisfied after removing B from R. Therefore, the order constraint of R is evolved to: R(A,C) $\subseteq$ (A > C).*

Similarly, when a relation is dropped by the user, then our algorithm will (1) drop all the type constraints of R from the MKB, and (2) drop the order constraint, if there is one.

5.2.2 Join Constraint Evolution

A join constraint $\mathcal{JC}$ is used to specify what is a meaningful way to combine information from possibly two ISs. When a primitive clause $(R_1.attr_i\ \theta\ R_2.attr_j)$ of a join constraint $\mathcal{JC}$ in the MKB is affected by del-attr(R_1, $attr_i$), our system does the following:

1. If the condition-replaceable parameter is set to false (β = false) and the condition-dispensable parameter is set to false (σ = false), then the join constraint is retracted from the MKB.
2. If the condition-replaceable parameter is set to false (β = false) and the condition-dispensable parameter is set to true (σ = true), then the join constraint would be rewritten with the affected primitive clause dropped.
3. If the condition-replaceable parameter is set to true (β = true), our algorithm will attempt to find an appropriate replacement in the same relation R_1. For example, if there is an attribute function-of constraint $R_1.attr_i = f(R_1.attr_x)$ with $attr_x$ an attribute of R_1 and $attr_i \neq attr_x$, then replace $(R_1.attr_i\ \theta\ R_2.attr_j)$ with $(f(R_1.attr_x)\ \theta\ R_2.attr_j)$ in $\mathcal{JC}$.
4. If our system fails to find a replacement within R_i and the condition-dispensable parameter is set to true (σ = true), then the join constraint can be rewritten with the affected primitive clause dropped. If the condition-dispensable parameter is set to false (σ = false), then the join constraint is retracted from the MKB.

When a relation R is dropped by the user, then our system will drop all the join constraints $\mathcal{JC}$ referencing to R, because new join constraints cannot be inferred by computing the transitive closure of existing join constraints in general.

5.2.3 Partial and Complete Information Constraint Evolution

A general partial/complete information constraint $\mathcal{PC}$ is given earlier in Equation (4). When an attribute $R_1.A$ is deleted from its IS, a $\mathcal{PC}$ between R_1 and another relation R_2 is affected if (1) $R_1.A$ appears in the projection list of the $\mathcal{PC}$, i.e., A $\in A_{i_1}, \cdots, A_{i_k}$, (2) $R_1.A$ appears in (one or more of the primitive clauses of) the selection condition of the $\mathcal{PC}$, i.e., A $\in A_{j_i}, \cdots, A_{j_l}$, (3) $R_1.A$ appears in both the projection list and the selection condition, i.e., (A $\in A_{i_1}, \cdots, A_{i_k}$) and (A $\in A_{j_i}, \cdots, A_{j_l}$) (see Equation 4).

1. If $R_1.A$ appears in the projection list only, we simply remove $R_1.A$ and the attribute corresponding to $R_1.A$ in the projection list of R_2, respectively, and the relationship between R_1 and R_2 stays the same. This can be proved by induction, because the selection conditions used on both sides of the $\mathcal{PC}$ stay the same.
2. If $R_1.A$ appears in the selection condition only, $\mathcal{PC}$ can be evolved by deleting the primitive clauses that reference $R_1.A$ from the selection list and with the new θ value set to $\supseteq$, if the old θ value was $\equiv$ or $\supseteq$. If the old θ value was $\subseteq$, then the old $\mathcal{PC}$ cannot be evolved and has to be dropped. The reason is that removing one or more primitive clauses from a selection condition may make the result set grow bigger in general. However, when the old θ value was $\subseteq$, i.e., the original fragment of R_1 is a subset of the fragment of R_2, removing one or more clauses from the selection condition for R_1 makes the fragment of R_1 bigger. We no longer know how to compare the new fragment of R_1 and the old fragment of R_2, so the partial/complete information condition between R_1 and R_2 has to be retracted from the MKB.
3. $R_1.A_{i_s}$ appears in both the projection list and the selection condition. This is a combination of the first and the second case. Algorithms presented earlier can be united to handle this case.

When a relation R is dropped by the user, then our system takes the following steps to evolve the affected partial/complete information constraints:

Partial/Complete Information Constraint Evolution Algorithm After Relation R is removed:

```
1. Find the set of affected partial/complete information constraints
   involving the relation R, and call this set Affected-PC.
2. Derive new partial/complete information constraints, New-PC,
   by computing the transitive closure of Affected-PC.
   That is, New-PC = Transitive-Closure - Affected-PC.
3. Add New-PC to MKB.
4. Remove the affected partial/complete information constraints, Affected-PC, from the MKB.
```

Example 8 *Assume there are two partial/complete information constraints in the MKB:*

$$\mathcal{PC}_{\boldsymbol{R},\boldsymbol{S}} = \left(\pi_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{R}) \supseteq \pi_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{S})\right) \ \textit{AND}\ \mathcal{PC}_{\boldsymbol{S},\boldsymbol{T}} = \left(\pi_{\boldsymbol{B}}(\boldsymbol{S}) \equiv \pi_{\boldsymbol{B}}(\boldsymbol{T})\right) \tag{9}$$

If the relation S is dropped by the user, then our system first finds both partial/complete information constraints, i.e., $\mathcal{PC}_{\boldsymbol{R},\boldsymbol{S}}$ and $\mathcal{PC}_{\boldsymbol{S},\boldsymbol{T}}$, are affected. Intersecting the projection lists of $\boldsymbol{S}$ of both $\mathcal{PC}$s, we derive a new partial/complete information constraint between R and T as:

$$\mathcal{PC}_{\boldsymbol{R},\boldsymbol{T}} = \left(\pi_{\boldsymbol{B}}(\boldsymbol{R}) \supseteq \pi_{\boldsymbol{B}}(\boldsymbol{T})\right) \tag{10}$$

Note that the new meta knowledge can be derived from the existing partial/complete information constraints, because the containment relationships of the existing $\mathcal{PC}$s are compatible (in our example, we have $\supseteq$ and $\equiv$). Therefore, we add this derived meta knowledge into the MKB and remove the two affected partial/complete information constraints given in Equation (9) from the MKB.

5.2.4 Attribute Function-Of Constraint Evolution

An *attribute function-of constraint* relates two attributes by defining a function that can transform one of them into another. Deleting an attribute or deleting a relation are the only capability changes that can affect the set of attribute function-of constraints in the MKB. Let Function-Of be the original set of attribute function-of constraints in the MKB. When an attribute A is deleted from an IS, Function-Of is evolved as follows:

Attribute Function-Of Constraint Evolution Algorithm After Attribute A Is Removed:

```
1. Find the set of affected attribute function-of constraints in the MKB, i.e.,
   that have A as the argument or as the result of the function-of constraints,
   and call this set Affected-Function-Of.
2. Derive new attribute function-of constraints, New-Function-Of,
   by computing the transitive closure of Affected-Function-Of.
   That is, New-Function-Of = Transitive-Closure - Affected-Function-Of.
3. Add New-Function-Of to the MKB.
4. Remove the affected attribute function-of constraint Affected-Function-Of from MKB.
```

Example 9 *Let's assume Function-Of =* $\{ R_i.A_r = f_1(R_j.A_s), R_j.A_s = f_2(R_k.A_t) \}$. *If* $R_j.A_s$ *is removed from its IS, both of the attribute function-of constraints in Function-Of are affected. Following the algorithm listed above, the transitive closure of the affected constraints is:*
Transitive-Closure = $\{ R_i.A_r = f_1(R_j.A_s), R_j.A_s = f_2(R_k.A_t), R_i.A_r = f_1 \circ f_2(R_k.A_t) \}$.
After adding the transitive closure to Function-Of and removing the affected function-of constraints, the set of attribute function-of constraints in the MKB contains one constraint only, namely the composed attribute function-of constraint – $R_i.A_r = f_1 \circ f_2(R_k.A_t)$.

When a relation R is deleted from its IS, for each of its attributes *attr* execute Attribute Function-Of Constraint Evolution Algorithm(*attr*) defined as above.

6 Formal Foundation for View Synchronization

In this section we give a formal definition of what constitutes a legal rewriting for a view definition VD which became obsolete after a capability change at an IS. The goal of our system is to use the information collected in the MKB to find acceptable (ideally equivalent) ways of rewriting a view definition VD so that the view is no longer undefined and all constraints imposed by the evolving parameters are satisfied by the new evolved view definition.

Consider a SELECT-FROM-WHERE view V defined as in Equation (6) by a query with a conjunction of primitive clauses in the WHERE clause. We assume in the rest of the paper that a view V is defined such that all *distinguished* attributes (i.e., the attributes used in the WHERE clause in an indispensable condition) are among the *preserved* attributes (i.e., the attributes in the SELECT clause) inheriting the evolving parameters from the condition they come from.

Definition 2 ***Affected View under Capability Change*** ch. *Let* ch *be a capability change of an underlying information source. The function* $DetectAffectedView(V, ch, MKB)$ *detects if the definition of the view* V *is affected by the change* ch *when the state of the knowledge base is* MKB. $DetectAffectedView(V, ch, MKB)$ *could have three outputs: (1) error - the view* V *is affected by the change* ch *but cannot be evolved because there exists at least a view component (i.e., attribute, relation or condition) affected by the change that is nondispensable* *and nonreplaceable (i.e, the attached dispensable parameter and replaceable parameter are both false). If such a component exists it is impossible to evolve the view and satisfy the evolving parameters in the same time. For example, if the change is drop attribute A, and attribute A is nondispensable and nonreplaceable in the* **SELECT** *clause of the view V, the view definition cannot be evolved such that the new definition satisfies evolving parameters for A); (2) true - the view V is affected by the change ch and it is possible to find a legal rewriting; note, that in this case, our system may still fail to evolve the view; (3) false - the view V is not affected by the change ch.*

When $DetectAffectedView(V, ch, MKB) = true$*, our view synchronization tool will try to evolve the view definition and possibly its content to be in sync with the new state of the information space, but it is not guaranteed that a legal evolved definition will be found. Definition 4 defines the notion of legal rewriting of the view V under capability change ch, and thus provides a formal foundation for the actions of the view synchronization module.*

In the following we use the term *view element* to refer to a *view component* such as an attribute, relation or condition used in the view definition together with the set of evolving parameters attached to it. If an attribute, relation or condition has no evolving parameters defined in the view definition, we assume that the view element corresponding to it has the default parameters as defined in Section 4.

Definition 3 *For a view V defined as in Equation (6), and a view V' having properties P1 to P6 from the Definition 4, the evolving parameters of V' are set as following:*

1. *For an element $X(par_1 = x_1, par_2 = x_2)$ of the view V' that is identical to the element X of the view V, the evolving parameters are left the same as in the original view definition.*
2. *For a new element $X(par_1 = x_1, par_2 = x_2)$ of the view V' that replaces exactly one element of the original view V, the evolving parameters are left the same. Note, that if an element of V is replaced by more than one new element of the view V', we say that each of the new elements replaces exactly one element of V.*
3. *For a new element $X(par_1 = x_1, par_2 = x_2)$ of the view V' that replaces more than one element of the original view definition $X_1(par_{1,1} = x_{1,1}, par_{1,2} = x_{1,2})$, ..., $X_k(par_{k,1} = x_{k,1}, par_{k,2} = x_{k,2})$[6], we set $X(par_1 = x_{1,1}\ AND\ \cdots\ AND\ x_{k,1},\ par_2 = x_{1,2}\ AND\ \cdots\ AND\ x_{k,2})$, as all $x_{j,i}, j = \overline{1,k}, i = \overline{1,2}$ are boolean.*
4. *The extent evolving parameter $\Delta_{V'}$ is equal to Δ_V.*

Definition 4 *A view V' is a* **legal rewriting** *of the view V under capability change ch if the following properties hold:*

- *P1. The view V' is* **no longer affected** *by the change ch, i.e.,* $DetectAffectedView(V', ch, MKB') = false$.
- *P2. The view V' can be* **evaluated** *in the new state of the information space. I.e., if the meta knowledge base MKB is changed into MKB' to reflect the change ch (see Section 5), then the view V' contains only elements defined in MKB' and can be evaluated given its definition.*

---

[6]Note that if some elements are replaced in the new definition, they are replaceable, i.e., second evolving parameter is *true*.

P3. The extent parameter Δ_V of V is satisfied by the view V'. That is, if $\bar{B}_V$ and $\bar{B}_{V'}$ are the attributes of interfaces of V and V', respectively, then

$$\pi_{\bar{B}_V \cap \bar{B}_{V'}}(V')\Delta_V \pi_{\bar{B}_V \cap \bar{B}_{V'}}(V) \tag{11}$$

is satisfied for any state of the underlying information sources.

*P4. All **evolving parameters** attached to the view elements such as attributes, relations or conditions of the view V, are satisfied by the view V'. For example, any legal rewriting of the view V must have in the interface all indispensable attributes (i.e., the ones having $\pi = true$).*

*P5. The definition of the view V' is **consistent** with the constraints of the MKB'. I.e., any new element (e.g., new condition in the **WHERE** clause) matches some constraint in MKB'. More precisely, new elements appear in the view V' only if they are required to replace existing elements that after a drop relation or drop attribute change must be replaced by other attributes. Thus, we have the following cases:*

1. *Case 1. A new element $f(S.A')(\pi = \pi_A, \varepsilon = \varepsilon_A)$ could appear in the **SELECT** clause of V' if it replaces exactly one element $R.A(\pi = \pi_A, \varepsilon = \varepsilon_A)$ from the **SELECT** clause of V[7]. The attributes $f(S.A')$ and $R.A$ must have the same type and the following condition holds:*

   $\exists R_1, R_2, \ldots, R_n$ having R and S among them, $\{\mathcal{JC}_{R_i,R_{i+1}} \mid \mathcal{JC}_{R_i,R_{i+1}}$ a join constraint in MKB', $i = \overline{1, n-1}\}$ such that $\exists R_{i_1}.A_{i_1}(= R.A), \ldots, R_{i_l}.A_{i_l}(= S.A')$, $\{i_1, \ldots, i_l\} \subseteq \{1, \ldots n\}$, $\exists \mathcal{F}_{R_{i_j}.A_{i_j}, R_{i_{j+1}}.A_{i_{j+1}}} = (R_{i_j}.A_{i_j} = f_j(R_{i_{j+1}}.A_{i_{j+1}}))$, for $j = \overline{1, l-1}$ function-of constraints and $f = f_1 \circ f_2 \circ \cdots \circ f_{l-1}$ a composition of the sequence of functions.

   That is, $R.A = f_1(f_2(\cdots f_{l-1}(S.A') \cdots))$ where the set of "linked" attributes $\{R_{i_1}.A_{i_1}(= R.A), \ldots, R_{i_l}.A_{i_l}(= S.A')\}$ are from the relation defined by the following expression:

   $$R_1 \bowtie_{\mathcal{JC}_{R_1,R_2}} R_2 \bowtie_{\mathcal{JC}_{R_2,R_3}} \cdots \bowtie_{\mathcal{JC}_{R_{n-1},R_n}} R_n \tag{12}$$

   *All relations from the expression (12) must appear in the **FROM** clause of the view V' and all primitive clauses from the set of join constraints used in expression (12) must appear in the **WHERE** clause of the view V'. Moreover, the attribute $R.A$ is replaced by $f(S.A')$ in V' in the conditions of the **WHERE** clause where $R.A$ appears in V.*

2. *Case 2. A new element $C(\sigma = \sigma_C, \beta = \beta_C)$ could appear in the **WHERE** clause of V' if it is a primitive clause in one of the sequences of join constraints used to replace an attribute $R.A$ as in Case 1.*

3. *Case 3. A new element $R(\alpha = \alpha_R, \theta = \theta_R)$ could appear in the **FROM** clause of V' if it appears in one of the sequences of joins used to replace an attribute $R.A$ as in Case 1.*

*P6. The definition of the view V' is **minimal** with respect to the set of the relations in the **FROM** clause and the set of conditions in the **WHERE** clause. That is, if we drop a relation from the **FROM** clause or a condition from the **WHERE** clause, the modified definition doesn't satisfy properties P1 to P5 any longer. P6 imposes that the new view definition cannot have extra elements that are not needed in the view, such as an extra attribute in the **SELECT** clause or an extra relation in the **FROM** clause that serves no purpose.*

[7] An attribute $R.A$ of V is to be replaced in the new view V' if for example the attribute is dropped from the relation R or the whole relation R is dropped.

P7. The evolving parameters for the elements of V' are defined as in Definition 3.

Example 10 *Let a view V be defined as follows:*

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & \textbf{\textit{Asia-Customer}}(\Delta = \supseteq) \textsf{ AS} \\
\textsf{SELECT} & \textbf{\textit{C.Name, C.Address}}(\pi = false, \varepsilon = true), \textbf{\textit{C.PhoneNo}} \\
\textsf{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\
\textsf{WHERE} & \textbf{\textit{(C.Name = F.Passenger) AND (F.Destination = 'Asia')}}
\end{array}
\tag{13}
$$

*And let's assume that change ch is "drop attribute **Address** from the relation **Customer**". We have to find a replacement for this attribute that could be obtained from a chain of join constraints defined in MKB'. Let's assume we have in MKB the following constraints:*

*(1) **Person** relation is defined by **Person(Name, SSN, Address)**;*

(2) $\mathcal{JC}_{\textbf{Customer, Person}} = (\textbf{Customer.Name} = \textbf{Person.Name})$;

(3) $\mathcal{F}_{\textbf{Customer.Address, Person.PermanentAddress}}$
$= (\textbf{Customer.Address} = \textbf{Person.PermanentAddress})$;

(4) $\mathcal{PC}_{\textbf{Customer, Person}} =$
$(\pi_{\textbf{Name, PermanentAddress}}(\textbf{Person}) \equiv \pi_{\textbf{Name, Address}}(\textbf{Customer}))$

*Equation (14) defining **Asia-Customer'** is a legal rewriting (new elements are underlined) of Equation (13):*

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & \textbf{\textit{Asia-Customer'}}\ (\Delta = \supseteq) \textsf{ AS} \\
\textsf{SELECT} & \textbf{\textit{C.Name}}, \underline{\textbf{\textit{P.PermanentAddress}}\ (\pi = false, \varepsilon = true)}, \textbf{\textit{C.PhoneNo}} \\
\textsf{FROM} & \textbf{\textit{Customer C, FlightRes F}}, \underline{\textbf{\textit{Person P}}} \\
\textsf{WHERE} & \textbf{\textit{(C.Name = F.Passenger) AND (F.Destination = 'Asia')}} \\
& \underline{\textbf{\textit{AND (P.Name = C.Name)}}}
\end{array}
\tag{14}
$$

*This legal rewriting uses the join constraint $\mathcal{JC}_{\textbf{Customer, Person}}$ to obtain the address from the relation **Person** by using the join relation $\left(\textbf{Customer} \bowtie_{\mathcal{JC}_{\textbf{Customer, Person}}} \textbf{Person}\right)$; and the function-of constraint defined by $\mathcal{F}_{\textbf{Customer.Address, Person.PermanentAddress}}$. Then the evolved view definition is given by Equation (14) which has all the properties P1 to P7, thus it is a legal rewriting. Note, that we can prove that for the evolved view defined by Equation (14), the extent parameter "$\supseteq$" is satisfied given the constraint (4), $\mathcal{PC}_{\textbf{Customer, Person}}$.*

Example 11 *Examples of queries that violate at least one of the properties from Definition 4 are given below:*
*(A) The evolving parameters from the initial query V are not satisfied by attribute **Address** being dropped, thus property P4 is violated:*

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & \textbf{\textit{Asia-Customer}}(\Delta = \supseteq) \textsf{ AS} \\
\textsf{SELECT} & \textbf{\textit{C.Name, C.PhoneNo}} \\
\textsf{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\
\textsf{WHERE} & \textbf{\textit{(C.Name = F.Passenger) AND (F.Destination = 'Asia')}}
\end{array}
\tag{15}
$$

*(B) The expression used to obtain a replacement for the attribute **Customer.Address** is not merged into the new definition by failing to add the join conditions in the WHERE clause, thus violating P5:*

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & \textbf{\textit{Asia-Customer}}(\Delta = \supseteq) \textsf{ AS} \\
\textsf{SELECT} & \textbf{\textit{C.Name}}, \underline{\textbf{\textit{P.PermanentAddress}}\ (\pi = false, \varepsilon = true)}, \textbf{\textit{C.PhoneNo}} \\
\textsf{FROM} & \textbf{\textit{Customer C, FlightRes F}}, \underline{\textbf{\textit{Person P}}} \\
\textsf{WHERE} & \textbf{\textit{(C.Name = F.Passenger) AND (F.Destination = 'Asia')}}
\end{array}
\tag{16}
$$

7 The Process of View Synchronization

As already discussed in Section 5, four of the six capability change operations we consider can be handled in a straightforward manner. Namely, the add-relation and add-attribute capability changes do not cause any changes to existing (and hence valid) views, and we assume that our current system will not further optimize existing views based on this new knowledge. The two rename capability change operators, rename-relation and rename-attribute, are caught by the name mapping service in the MKB and hence also do not require any synchronization at the individual view site level.

However, the two remaining capability change operators, i.e., delete-attribute and delete-relation, cause existing views to become invalid and hence need to be addressed by the view synchronization algorithm. Below, we present the algorithm for handling the most difficult operator, namely, the delete-relation operator, in depth. The algorithm for the delete-attribute operator is a simplified version of the delete-relation algorithm given below, and hence is omitted in this paper due to space limitations.

Given delete-relation(R), we assume for the following that the view query V uses R only once in the FROM clause. The algorithm ***LegalRewritings*** could be easily adapted for a more general case when the relation R appears more than once in the FROM clause. We also assume that any join constraint in MKB is augmented with the order constraints defined for the relations involved in that join constraint. We start by giving some definitions for the concepts needed for the view synchronization process.

Example 12 *The following query will be used to illustrate the steps for rewriting under a delete relation change "delete relation* ***Customer****". The view* ***Customer-Passengers-Asia*** *defines pairs (passenger, participant) of passengers flying to Asia and participants to a tour in Asia that fly and start the tour at the same day, respectively. Such a view could be used to see what participants to a tour are flying to "Asia" at the same day as the tour starts.*

CREATE VIEW	***Customer-Passengers-Asia*** (Δ_V) *AS*
SELECT	***C.Name*** $(\pi = false, \epsilon = true)$, ***C.Age*** $(\pi = true, \epsilon = true)$,
	P.Participant $(\pi = true, \epsilon = true)$, ***P.TourID*** $(\pi = true, \epsilon = true)$
FROM	***Customer C*** $(\alpha = true, \theta = true)$, ***FlightRes F*** $(\alpha = true, \theta = true)$,
	Participant P $(\alpha = true, \theta = true)$
WHERE	***(C.Name = F.Passenger)*** $(\sigma = false, \beta = true)$ ***AND (F.Destination = 'Asia') AND***
	(P.StartingDate = F.Date) AND (P.Location = 'Asia')

(17)

Definition 5 ***Meta Knowledge Base Hypergraph*** $\mathcal{H}$***(MKB).*** *Given the meta knowledge base MKB, we define the hypergraph associated with it* $\mathcal{H}(MKB)$*, of the set of relations* $\{R_1, \ldots, R_n\}$ *described in MKB, the set of join constraints* $\{\mathcal{JC}_1, \ldots, \mathcal{JC}_m\}$ *in MKB, and the set of function-of constraints* $\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ *in MKB, by representing the relations and function-of constraints by hyperedges, and the attributes and join constraints by nodes. A function-of constraint is a hyperedge connecting only two attribute nodes, thus, for simplicity, we represent it by a direct edge connecting the corresponding attributes (depicted by arrows in Figure 6). Two relations share a constraint node (depicted by* ***JCi*** *in Figure 6) if and only if the node corresponds to a join constraint defined for the two relations. Figure 6 depicts the hypergraph associated to Example 2.*

Definition 6 ***Connected Sub-Hypergraph*** $\mathcal{H}_R$***(MKB).*** *For a relation R, we define the sub-hypergraph of $\mathcal{H}$(MKB) that contains R as following:*

$$\mathcal{H}_R(MKB) = \{\mathcal{S}_R(MKB), \mathcal{J}_R(MKB), \mathcal{A}_R(MKB), \mathcal{F}_R(MKB)\} \tag{18}$$

$\mathcal{S}_R(MKB) = \{S_1, S_2, \ldots, S_k\}$ is a set of relations of $\mathcal{H}(MKB)$ containing R, such that for any two relations S_1 and S_2 in $\mathcal{S}_R(MKB)$, there exists a sequence of join constraints $\mathcal{JC}_{S_1,R_1}, \ldots, \mathcal{JC}_{R_n,S_2}$ defined in MKB, with $R_1, \ldots, R_n \in \mathcal{S}_R(MKB)$. I.e., the following relation is a meaningful way (given MKB) to combine the relations S_1 and S_2: $S_1 \bowtie_{\mathcal{JC}_{S_1,R_1}} R_1 \cdots \bowtie \cdots \bowtie_{\mathcal{JC}_{R_n,S_2}} S_2$.
$\mathcal{J}_R(MKB) = \{\mathcal{JC}_{S,S'} \mid S, S' \in \mathcal{S}_R(MKB)\}$ is the set of join constraints in MKB defined between relations in $\mathcal{S}_R(MKB)$,
$\mathcal{A}_R(MKB) = \{A \mid A \in S, S \in \mathcal{S}_R(MKB)\}$ is the set of attributes of the relations in $\mathcal{S}_R(MKB)$, and
$\mathcal{F}_R(MKB) = \{\mathcal{F}_{A,B} \mid A, B \in \mathcal{A}_R(MKB)\}$ is the set of function-of constraints defined between relations in $\mathcal{S}_R(MKB)$.

Example 13 *Figure 6 depicts two connected components for the hypergraph $\mathcal{H}$(MKB) for Example 1. In Figure 6 for $R =$* ***Customer*** *we have $\mathcal{S}_{\textbf{Customer}}$(MKB) = {* ***FlightRes, Customer, Participant, Tour, Accident–Ins*** *}.*

Definition 7 ***R-mapping of a view query V into sub-hypergraph $\mathcal{H}_R(MKB)$.*** *For a view query V defined by an expression (6) and a relation R from the* FROM *clause of the view query V, we define the R-mapping of the query into $\mathcal{H}_R(MKB)$ by R-mapping(V, $\mathcal{H}_R(MKB)$) $= (Max(V_R), Min(\mathcal{H}_R))$ to be a pair of two subexpressions one constructed from the view query V and one constructed from the connected sub-hypergraph $\mathcal{H}_R(MKB)$ such that the followings hold:*
(I) The expression $Max(V_R)$ is of the form:

$$Max(V_R) \;=\; R_{v_1} \bowtie_{\mathcal{C}_{R_{v_1},R_{v_2}}} \cdots \bowtie_{\mathcal{C}_{R_{v_{l-1}},R_{v_l}}} R_{v_l} \tag{19}$$

such that relations $\{R_{v_1}, \ldots, R_{v_l}\}$ are from the FROM *clause of V, $R \in \{R_{v_1}, \ldots, R_{v_l}\}$, and $\{\mathcal{C}_{R_{v_1},R_{v_2}}, \ldots, \mathcal{C}_{R_{v_{l-1}},R_{v_l}}\}$ are conjunction of primitive clauses from the* WHERE *clause of V. A conjunction $\mathcal{C}_{R_{v_{s-1}},R_{v_s}}$ contains all the primitive clauses that use attributes of relations $R_{v_{s-1}}$ and R_{v_s} (both local and join conditions).*
(II) The expression $Min(\mathcal{H}_R)$ is of the form:

$$Min(\mathcal{H}_R) = R_{h_1} \bowtie_{\mathcal{JC}_{R_{h_1},R_{h_2}}} \cdots\cdots \bowtie_{\mathcal{JC}_{R_{h_{s-1}},R_{h_s}}} R_{h_s} \tag{20}$$

where relations $\{R_{h_1}, \ldots, R_{h_s}\} \subseteq \mathcal{S}_R(MKB)$, $R \in \{R_{h_1}, \ldots, R_{h_s}\}$ and $\{\mathcal{JC}_{R_{h_1},R_{h_2}}, \ldots, \mathcal{JC}_{R_{h_{s-1}},R_{h_s}}\} \subseteq \mathcal{J}_R(MKB)$.
(III) The relation defined by $Max(V_R)$ is contained in the relation defined by $Min(\mathcal{H}_R)$:

$$\left(R_{v_1} \bowtie_{\mathcal{C}_{R_{v_1},R_{v_2}}} \cdots \bowtie \cdots \bowtie_{\mathcal{C}_{R_{v_{l-1}},R_{v_l}}} R_{v_l}\right) \subseteq \left(R_{h_1} \bowtie_{\mathcal{JC}_{R_{h_1},R_{h_2}}} \cdots \bowtie \cdots \bowtie_{\mathcal{JC}_{R_{h_{s-1}},R_{h_s}}} R_{h_s}\right) \tag{21}$$

(IV) The expression $Max(V_R)$ is maximal with the properties (I) and (III). I.e., there is no other relations from the FROM *clause of the view V that could be added to it and still be able to find a subexpression in $\mathcal{H}_R(MKB)$ such that (I) and (III) are satisfied.*

The expression $Min(\mathcal{H}_R)$ is minimal with the properties (II) and (III). I.e., we cannot drop a relation from it and still have (II) and (III) satisfied.

Algorithm for Computing R-mapping(V, $\mathcal{H}_R(MKB)) = (Max(V_R), Min(\mathcal{H}_R))$.
To find the expressions $Max(V_R)$ and $Min(\mathcal{H}_R)$ with the property (III), it is sufficient ([Ull89]) to have that:
(M1) $\{R_{h_1}, \ldots, R_{h_s}\} \subseteq \{R_{v_1}, \ldots, R_{v_l}\}$;
(M2) each join constraint $\mathcal{JC}_{S,S'}$ of expression $Min(\mathcal{H}_R)$ (20) is implied by the corresponding join condition $\mathcal{C}_{S,S'}$ of expression $Max(V_R)$ (19), where $S, S' \in \{R_{h_1}, \ldots, R_{h_s}\}$.

To find $Max(V_R)$ in V having property (M2), we start by selecting all relations that join with R in V with a join condition $\mathcal{C}_{R,S}$ such that $\exists \mathcal{JC}_{R,S}$ in MKB, and $\mathcal{C}_{R,S}$ implies $\mathcal{JC}_{R,S}$. Then for any relation found by this first step, we recursively find others that are joined with it with join conditions that imply the corresponding join constraints in MKB, until we cannot find any new relation to add. $Min(\mathcal{H}_R)$ is obtained from $Max(V_R)$ by dropping the join conditions that are not part of the join constraints defined in MKB.

Example 14

*The maximal subexpression $Max($**Customer-Passenger-Asia**$_{\textbf{Customer}})$ of the query (17) and change $ch =$ **delete−relation Customer** is:*

$$Max(\textit{\textbf{Customer-Passenger-Asia}}_{\textbf{\textit{Customer}}}) = \textbf{\textit{FlightRes}} \bowtie_{\left(\substack{(FlightRes.Passenger\ =\ Customer.Name) \\ AND\ (FlightRes.Destination\ =\ 'Asia')}\right)} \textbf{\textit{Customer}} \tag{22}$$

In the Figure 6, the minimal subexpression $Min(\mathcal{H}_{\textbf{Customer}})$ of the $\mathcal{H}_{\textbf{Customer}}(MKB)$ is marked by bold lines and corresponds to:

$$Min(\mathcal{H}_{\textbf{\textit{Customer}}}) = \textbf{\textit{FlightRes}} \bowtie_{\underbrace{(FlightRes.Passenger\ =\ Customer.Name)}_{\textbf{\textit{JC1}}}} \textbf{\textit{Customer}} \tag{23}$$

The relation defined by expression (22) is contained in the relation defined by expression (23) and they are maximal and minimal, respectively, with this property. I.e., the expression (22) is the maximal subexpression in the view query (17) having the properties (I) to (III): if we add a new relation from the query (17) to it, we cannot find any longer a subexpression of $\mathcal{H}(MKB)$ so that these properties hold. The same we can say for the subexpression (23) to be minimal with the properties (I) to (III).

Definition 8 ***R-replacement***$(V, \mathcal{H}_R(MKB))$*. For a given view definition V and the MKB, we compute a set of expressions constructed from $\mathcal{H}_R(MKB)$ that doesn't contain R and could be used to meaningfully replace the maximal subexpression $Max(V_R)$ in V. Let MKB' be the meta knowledge base evolved from MKB (Section 5) when relation R is dropped; and $\mathcal{H}'_R(MKB')$ be the sub-hypergraph of $\mathcal{H}_R(MKB)$ obtained by erasing hyperedge R. We define R-replacement$(V, \mathcal{H}_R(MKB)) = \{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ to be a set of subexpressions constructed from $\mathcal{H}'_R(MKB')$ and $Max(V_R)$.*
A subexpression $Max(V_{1,R})$ has the following properties:
(I) $Max(V_{1,R}) = R_1 \bowtie_{\mathcal{JC}_{R_1,R_2}} \cdots \bowtie_{\mathcal{JC}_{R_{k-1},R_k}}$ with $R_1, \ldots, R_k$ and $\mathcal{JC}_{R_1,R_2}, \ldots, \mathcal{JC}_{R_{k-1},R_k}$ in $\mathcal{H}'_R(MKB')$.
(II) R doesn't appear in $Max(V_{j,R})$. I.e., R not among $R_1, \ldots, R_k$.

(III) The expression $Min(\mathcal{H}_R)$ without R, $Min(\mathcal{H}'_R)$, could be mapped into $Max(V_{j,R})$. That is, if $Min(\mathcal{H}_R)$ is given by the equation (20) then: $\{R_{h_1}, \ldots, R_{h_s}\} \setminus \{R\} \subseteq \{R_1, \ldots, R_k\}$ and $\{\mathcal{JC}_{R_{h_1}, R_{h_2}}, \ldots \mathcal{JC}_{R_{h_{s-1}}, R_{h_s}}\}$ $\setminus \{\mathcal{JC}_{S,S'} \mid S = R \text{ or } S' = R\} \subseteq \{\mathcal{JC}_{R_1,R_2,\ldots \mathcal{JC}_{R_{k-1}}, R_k}\}$.

(IV) For any attribute $A \in R$ that is a nondispensable attribute in the query view V, the expression $Max(V_{j,R})$ contains a relation $S \in \{R_1, \ldots, R_k\}$ such that there exists a function-of constraint $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$ in MKB. We call the relation S a ***cover*** *for the attribute A and the attribute $f(S.B)$ a* ***replacement*** *for the attribute A in $Max(V_{j,R})$.*

Algorithm for computing R-replacement$(V, \mathcal{H}_R(MKB))$.

Erasing R from the connected sub-hypergraph $\mathcal{H}_R(MKB)$ could lead to a disconnected sub-hypergraph $\mathcal{H}'_R(MKB')$. If $\mathcal{H}'_R(MKB')$ is disconnected and the relations left in $Min(\mathcal{H}'_R)$ are in disconnected components then the set R-replacement$(V, \mathcal{H}_R(MKB))$ is empty.

Example 15 *In Figure 7, the expression $Min(\mathcal{H}'_{\textbf{Customer}})$ corresponding to the subexpression (23) is marked with bold lines: $Min(\mathcal{H}'_{\textbf{Customer}}) = \sigma_{\textbf{FlightRes.Destination = 'Asia'}}(\textbf{FlightRes})$.*

If relations left in $Min(\mathcal{H}'_R)$ are in a connected component of $\mathcal{H}'_R(MKB')$, we construct the set $\{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ in two steps:

Step 1. For any nondispensable attribute A of R, from the SELECT clause, we find a set of pairs (relation, function-of), $Cover(A)$, from $\mathcal{H}'_R(MKB')$ such that $\forall (S, (R.A = f(S.B))) \in Cover(A)$, S is in $\mathcal{H}'_R(MKB')$ and there exists a function-of constraint $\mathcal{F}_{R.A,S.B}$ in MKB such that $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$.

If there exists a nondispensable attribute A of R such that $Cover(A) = \emptyset$, then the set R-replacement$(V, \mathcal{H}_R(MKB))$ is empty.

Example 16 *In our example, the only nondispensable attribute is* ***Customer.Name****. Using the hypergraph depicted in Figure 7, we find: $Cover($**Customer.Name**$) =$*
{ (***Accident–Ins****,* ***(Customer.Name = Accident–Ins.Holder)****),*
(***Participant****,* ***(Customer.Name = Participant.Participant)*** *),*
(***FlightRes****,* ***(Customer.Name = FlightRes.Passenger)*** *) }.*

Step 2. At this step we build expressions by joining (using join constraints from $\mathcal{H}'_R(MKB')$) $Min(\mathcal{H}'_R)$ with relations from $Cover(A)$, for each nondispensable attribute A of R. If the expression obtained is a "connected" expression (i.e., corresponds to a connected sub-hypergraph of $\mathcal{H}'_R(MKB)$), then we found an expression $Max(V_{j,R})$.

Example 17 *For our example and the set $Cover($**Customer.Name**$)$, let's construct the candidate expressions $Max($**Customer-Passenger-Asia**$_{j,\textbf{Customer}})$ and define what is the replacement for the attribute* ***Customer.Name****.*

(1) For (***Accident–Ins****,* ***(Customer.Name = Accident–Ins.Holder)****) $\in Cover($**Customer.Name**$)$, we apply Step 2.:*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{1,\textbf{\textit{Customer}}}) = \underbrace{\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'\textbf{\textit{Customer}})} \bowtie \left(\underbrace{\underbrace{\textit{(FlightRes.Passenger = Accident-Ins.Holder)}}_{JC6} \ AND\ \textit{FlightRes.Destination = 'Asia'}}_{Min(\mathcal{H}'\textbf{\textit{Customer}})} \right) \underbrace{\textbf{\textit{Accident-Ins}}}_{in\ Cover(\textbf{\textit{Customer.Name}})} \tag{24}$$

*(2) For (**Participant**, (**Customer.Name** = **Participant.Participant**))* $\in Cover($**Customer.Name**$)$, *we apply Step 2.:*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{2,\textbf{\textit{Customer}}}) = \underbrace{\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'\textbf{\textit{Customer}})} \bowtie \underbrace{\textbf{\textit{Participant}}}_{Cover(\textbf{\textit{Customer.Name}})} \tag{25}$$

*We see that for this particular cover for the attribute **Customer.Name**, we fail to find a connected expression, as there is no join constraint between the relations **FlightRes** and **Participant** in MKB' (Figure 7). Then we cannot generate any replacement for* $Max(\textbf{\textit{Customer-Passenger-Asia}}_{\textbf{\textit{Customer}}})$ *using this cover.*

*(3) For (**FlightRes**, (**Customer.Name** = **FlightRes.Passenger**))* $\in Cover($**Customer.Name**$)$, *we apply Step 2.:*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{3,\textbf{\textit{Customer}}}) = \underbrace{\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'\textbf{\textit{Customer}})} \bowtie \underbrace{\textbf{\textit{FlightRes}}}_{in\ Cover(\textbf{\textit{Customer.Name}})} \tag{26}$$

*We assume that any relation could be joined with itself, then expression (26) is equivalent to **FlightRes**, i.e.,*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{3,\textbf{\textit{Customer}}}) = \underbrace{\sigma_{\textit{(FlightRes.Destination = 'Asia')}}\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'\textbf{\textit{Customer}}),in\ Cover(\textbf{\textit{Customer.Name}})} \tag{27}$$

Now we are ready to give a **LegalRewriting** algorithm that has as input a view query V, the meta knowledge base MKB and a change "delete relation R" and returns all possible rewritings of the view V.

Algorithm for Finding Legal Rewritings: LegalRewritings(V, ch **=delete−relation** R**, MKB, MKB')**

INPUT:

the view definition V defined as in Equation 6;

the change ch = *delete-relation* R;

the augmented MKB represented by the hypergraph $\mathcal{H}(MKB)$

the evolved augmented MKB' represented by the hypergraph $\mathcal{H}(MKB')$

OUTPUT:

A set of legal rewritings $V_1, \ldots V_l$ of V.

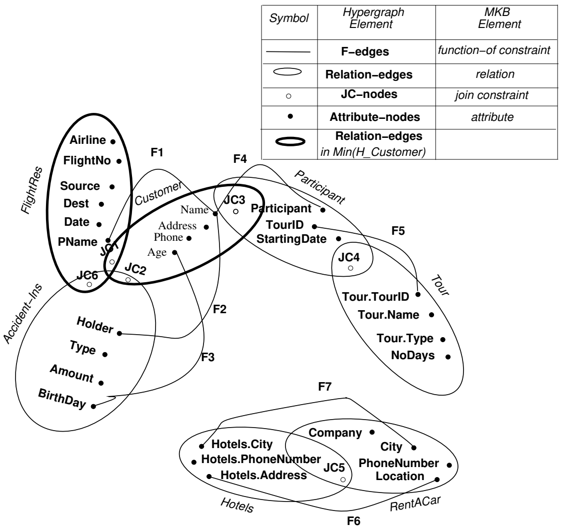

Figure 6: The Original Hypergraph $\mathcal{H}$(MKB) for Example 1 and View (17).

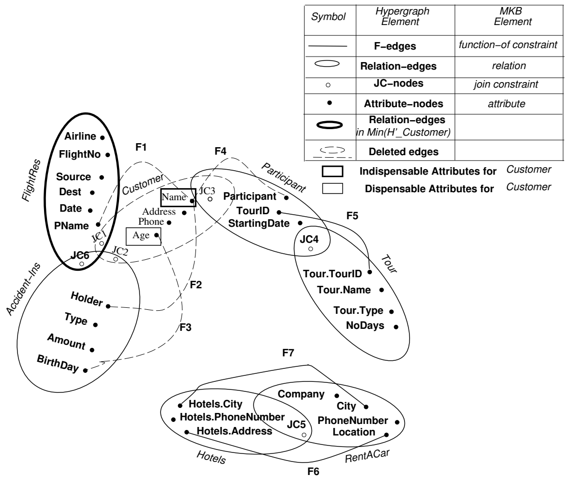

Figure 7: The Evolved Hypergraph $\mathcal{H}$(MKB') for Example 1 and View (17).

Step 1. Construct the sub-hypergraph $\mathcal{H}_R(MKB)$ as defined in Definition 6.

Step 2. Compute R-mapping(V, $\mathcal{H}_R(MKB)) = (Max(V_R), Min(\mathcal{H}_R))$ as defined in Definition 7.

Step 3. Compute R-replacement(V, $\mathcal{H}'_R(MKB')) = \{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ as defined in Definition 8. If R-replacement(V, $\mathcal{H}'_R(MKB') = \emptyset$ then the algorithm fails to find an evolved view definition for the view V.

Step 4. An evolved query V' is found by replacing $Max(V_R)$ with $Max(V_{j,R})$ and substitute the attributes of R in V with the corresponding replacements found in $Max(V_{j,R})$. Because some more conditions are added in the WHERE clause (corresponding to the join conditions in $Max(V_{j,R})$), we have to check if there are no inconsistency in the WHERE clause. Example 18 below gives some examples of evolved view definitions for view defined by the Equation (17).

Step 5. Set the evolving parameters for all V' obtained at Step 4, as defined in Section 6, Definition 4.

Step 6. All the rewritings from Step 4 have properties P1, P2, P4, P5, P6, and P7 from Definition 4, Section 6. At this step, we have to check for which rewriting V' obtained in Step 4 the extent parameter Δ_V of the query V is satisfied in order to see if the property P3 from Definition 4 is satisfied. This problem is similar to the problem of writing a query using views which was extensively studied in the database community [CKP95, LSK95]. However, in our problem domain, we have an added issue of the availability of the set of partial/complete information constraints defined in MKB' that could be used to compare the extent of the initial view V and the extent of the evolved view V'. This development is beyond the scope of current paper, but we plan, in the future work, to address this problem.

Example 18 *For our view **Customer-Passenger-Asia** defined by the Equation (17), let's apply Steps 4 and 5 from the algorithm and find replacement under the change "delete relation **Customer**".*

The expression $Max(\textbf{Customer-Passenger-Asia}_{\textbf{Customer}}) =$ $\textbf{FlightRes} \bowtie_{\left(\begin{array}{c}(FlightRes.Passenger\ =\ Customer.Name)\ AND \\ (FlightRes.Destination\ =\ 'Asia')\end{array}\right)} \textbf{Customer}$ *could be replaced by one of the following expressions found at Step 3 of the **LegalRewritings** algorithm:*

(1) $Max\,\textbf{\textit{Customer-Passenger-Asia}}_{1,}\textbf{\textit{Customer}})$ = $\textbf{\textit{FlightRes}} \bowtie_{\substack{((\textit{FlightRes.Passenger} = \textit{Accident-Ins.Holder})) \\ AND\ \textit{FlightRes.Destination} = \textit{'Asia'}}}$ ***Accident–Ins***.

For this particular case, we see that the attribute ***Customer.Age*** *is also covered by the relation* ***Accident–Ins:*** *(****Customer.Age*** $=$ *(today* $-$ ***Accident–Ins.Birthday****)/365). In this case, we can replace* ***Customer.Age*** *in the view, too. A new rewriting of Equation (17) using this substitution is given by Equation (28). There are no contradictions in the* WHERE *clause after the replacement is done.*

$$
\begin{array}{ll}
\textsf{\textit{CREATE VIEW}} & \textbf{\textit{Customer-Passengers-Asia}}\ \textit{AS} \\
\textsf{\textit{SELECT}} & \underline{\textbf{\textit{AI.Name}}\ (\pi = false, \epsilon = true)},\ \underline{\textbf{\textit{f(AI.BirthDay)}}\ (\pi = true, \epsilon = true),} \\
 & \underline{\textbf{\textit{P.Participant}}\ (\pi = true, \epsilon = true),\ \textbf{\textit{P.TourID}}\ (\pi = true, \epsilon = true)} \\
\textsf{\textit{FROM}} & \underline{\textbf{\textit{Accident-Ins AI}}\ (\alpha = true, \theta = true)},\ \textbf{\textit{FlightRes F}}\ (\alpha = true, \theta = true), \\
 & \underline{\textbf{\textit{Participant P}}\ (\alpha = true, \theta = true)} \\
\textsf{\textit{WHERE}} & \underline{\textbf{\textit{(AI.Holder = F.Passenger)}}\ (\sigma = false, \beta = true)}\ \textbf{\textit{AND (F.Destination = 'Asia') AND}} \\
 & \underline{\textbf{\textit{(P.StartingDate = F.Date) AND (P.Location = 'Asia')}}}
\end{array}
\tag{28}
$$

(2) $Max(\textbf{\textit{Customer-Passenger-Asia}}_{3,}\textbf{\textit{Customer}}) = \sigma_{(\textit{FlightRes.Destination} = \textit{'Asia'})}\textbf{\textit{FlightRes}}$.

A new rewriting of the query (17) is given by the query (29). There are no contradictions in the WHERE *clause after the replacement is done.*

$$
\begin{array}{ll}
\textsf{\textit{CREATE VIEW}} & \textbf{\textit{Customer-Passengers-Asia}}\ \textit{AS} \\
\textsf{\textit{SELECT}} & \underline{\textbf{\textit{F.Passenger}}\ (\pi = false, \epsilon = true),} \\
 & \overline{\textbf{\textit{P.Participant}}\ (\pi = true, \epsilon = true)},\ \textbf{\textit{P.TourID}}\ (\pi = true, \epsilon = true) \\
\textsf{\textit{FROM}} & \textbf{\textit{FlightRes F}}\ (\alpha = true, \theta = true), \\
 & \textbf{\textit{Participant P}}\ (\alpha = true, \theta = true) \\
\textsf{\textit{WHERE}} & \textbf{\textit{(F.Destination = 'Asia') AND}} \\
 & \textbf{\textit{(P.StartingDate = F.Date) AND (P.Location = 'Asia')}}
\end{array}
\tag{29}
$$

8 Related Work

This work addresses issues of view evolution caused by capability changes in participating information sources. To our knowledge, this problem has not been studied before in the database literature. However, some subproblems that must be solved in the context of our DVS system were studied before, most of them in the area of information integration.

The Dynamic Information Integration Model (DIIM) we propose in a University of Michigan Digital Library Project [NR96, NR97] is a model that allows information sources to dynamically participate in an information integration system. The DIIM query language allows loosely specified queries that the DIIM system refines into executable, well-defined queries based on the capability descriptions each information source exports when joining the DIIM system. Issues of defining evolving view models, MKB evolution process, view synchronization, etc., are not discussed. In the DVS system, on the other hand, we solve the problem of how to change well-defined view queries when the capabilities of the underlying ISs change.

In the work of Levy et al. [LSK95], a global information system is designed using the world-view approach where the external information sources are described relative to the unified world-view relations. The queries in the global information system are expressed in terms of world-view relations, thus the system must "translate" them into queries expressed using the base relations exported by external information sources. This work addresses the problem of choosing the right base relations for query executions, a process that is similar to the DVS process of redefining the view using "appropriate" relations or attributes based on DVS constraints. However, evolution of views as handled in our current paper is not discussed at all in [LSK95].

Papakonstantinou et al. [PGMW95] are pursuing the goal of information gathering across multiple sources. Their proposed language OEM assumes queries that explicitly list the source identifiers of the database from which the data is to be taken. Their data model allows information sources to describe their capabilities as well, but they don't assume that these capabilities could be changed and thus they do not address the view synchronization problem. The latter is the problem we address in our work.

In a separate project, we [RR95, RRL97] use view technology to handle schema changes transparently in a centralized environment. In the TSE framework, a user works on special-tailored view schemas instead of working on the base schema directly. Schema changes then are specified against the view schema (whereas in DVS they are triggered due to changes of ISs). The TSE system is responsible for deriving an alternate view schema to simulate the effects of schema evolution while preserving the current view schemas. When the schema change is capacity-reducing/preserving, TSE derives the target view schema from the original view schema. When the schema change is capacity-augmenting, TSE first translates the schema change request specified on the view schema into an in-place schema change specified on the base schema, augments the base schema, derives the target view schema based on the augmented base schema, and finally restores the original base schema by applying an inverse schema change transformation on the augmented base schema. In both cases, none of the existing view schemas are affected, since in both cases the original base schema is preserved. Furthermore, all view schemas are kept in the system. In DVS system, instead we consider schema (capability) changes coming from the information space, and we study the effects of such IS capability changes on existing view's definition. If all of the ISs were willing to preserve the relations that are referenced by the view definitions (i.e., no schema changes were to happen on base relations), then DVS behaves like TSE.

Gupta et al. [GMR95] and Mohania et al. [MD96] address the problem of how most efficiently to update view tuples after a view redefinition takes place. And, they study under which conditions this view update can take place without requiring access to base relations, i.e., the self-maintainability issue. Their algorithms could potentially be applied to views in the context of our overall framework, once DVS has determined an acceptable view redefinition - though we expect that adjustments to their solutions must be made to account for when one does not have accesses to old, dropped data.

9 Conclusion

Our work is the first to study the problem of view evolution in a dynamic environment. In our system, views survive even when the underlying ISs upon which they are defined change their capabilities. One component of our solution approach is for a user to specify evolution criteria as part of the view specification. In order to find alternate replacements for components of a view affected by IS capability changes, we have developed a description model for capturing the capabilities of ISs as well as interrelationships between ISs.

Equipped with the extended view definition language and the IS description model, we then propose strategies for the view synchronization process. First, we introduce algorithms for MKB evolution. Then, we identify which views are potentially affected by the MKB evolution. Thereafter, based on the view change semantics expressed by our view definition language, our view synchronizer explores alternate techniques for view rewriting with the goal of meeting all view preservation constraints in VD, extracting appropriate information from other ISs as replacement of the modified capabilities using the MKB, and then generating new view definitions. The ideas of *view synchronization* are illustrated in this paper by describing algorithms for evolving view definitions caused by the 'delete-relation" capability change.

To summarize, the main contributions of this paper are:

- We have formally presented a set of properties for a view rewriting to be legal when a view definition became obsolete after a capability change at an IS. In this paper, we investigate in particular complex view rewriting possibilities through multiple join constraints given in MKB.
- Since our work discuss applications operated in a dynamic environment, issues associated with MKB management are important. For that, we have presented the MKB evolution algorithm for the basic set of capability changes.
- To demonstrate our solution approach, we have presented the algorithm for handling the most difficult capability change operator, namely, the delete-relation operator, in depth.

This work has opened a new problem domain, and much future research could be conducted within the context of our proposed framework, such as appropriate cost models for maximal view preservation, etc.

References

[CKP95] S. Chaudhuri, R. Krishnamurthy, and S. Potamianos. Optimizing Query with Materialized Views. In *Proceedings of IEEE International Conference on Data Engineering*, 1995.

[GMR95] A. Gupta, I.S. Mumick, and K.A. Ross. Adapting Materialized Views after Redefinition. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 1995.

[Huy96] N. Huyn. Efficient View Self-Maintenance. *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, June 1996.

[LNR97a] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping Virtual Information Resources Up and Running. In *Proceedings of IBM Centre for Advanced Studies Conference CASCON97, Best Paper Award*, pages 1–14, November 1997.

[LNR97b] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Framework: View Evolution in an Evolving Environment. Technical Report WPI-CS-TR-97-4, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[LSK95] A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems. Special Issue on Networked Information Discovery and Retrieval*, 1995.

[MD96] M. Mohania and G. Dong. Algorithms for Adapting Materialized Views in Data Warehouses. *International Symposium on Cooperative Database Systems for Advanced Applications*, December 1996.

[NR96] A. Nica and E. A. Rundensteiner. The Dynamic Information Integration Model. Technical Report CSE-TR-311-96, University of Michigan, Ann Arbor, EECS Dept. CSE Division, 1996.

[NR97] A. Nica and E. A. Rundensteiner. On Translating Loosely-Specified Queries into Executable Plans in Large-Scale Information Systems. In *Proceedings of Second IFCIS International Conference on Cooperative Information Systems CoopIS'97*, pages 213–222, June 1997.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[RLN97] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

[RR95] Y. G. Ra and E. A. Rundensteiner. A Transparent Object-Oriented Schema Change Approach Using View Schema Evolution. In *Proceedings of IEEE International Conference on Data Engineering*, pages 165–172, March 1995.

[RRL97] E. A. Rundensteiner, Y. G. Ra, and A. J. Lee. Transparent Schema Evolution (TSE) Using Object-Oriented View Technology: Taking a Fresh Look. Technical Report WPI-CS-TR-97-3, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[Ull89] J.D. Ullman. *Principle of Database and Knowledge-Base Systems.* Computer Science Press, 1989.

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of International Conference on Information and Knowledge Management*, pages 25–30, November 1995.