On Preserving Views In Evolving Environments *

Elke A. Rundensteiner
Dept. of CS, WPI
Worcester, MA 01609
rundenst@cs.wpi.edu

Amy J. Lee
Dept. of EECS, U of Mich.
Ann Arbor, MI 48105
amylee@eecs.umich.edu

Anisoara Nica
Dept. of EECS, U of Mich.
Ann Arbor, MI 48105
anica@eecs.umich.edu

Abstract

The construction and maintenance of data warehouses (views) in large-scale environments composed of numerous distributed information sources (ISs) such as the WWW has received great attention recently. Such environments are plagued with continuously changing information because ISs tend to continuously evolve by modifying not only their content but also their query capabilities and interface and by joining or leaving the environment at any time. In this paper, we outline our position on issues related to the challenging new problem of how to adapt views in such evolving environments. We first present a taxonomy of view adaptation problems by describing the dimensions along which view adaptation problems can be classified. Based on this taxonomy, we identify a new view adaptation problem for view evolution in the context of ISs capability changes, which we call *View Synchronization*. We also outline the Evolvable View Environment (EVE) that we propose as framework for solving the view synchronization problem, along with our decisions concerning some of the key design issues surrounding EVE.

*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 and Informix.

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997

(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

1 Problem Description

Advanced applications such as web-based information services, data warehousing, digital libraries, and data mining often specify views on a large number of dynamic information sources (ISs) [Wid95]. There is typically a large variety and number of ISs in these environments, each modeled by diverse data models and each supporting different query languages and interfaces. Furthermore, individual ISs freely update both their content and their capabilities[1]. Lastly, they can even join or leave the environment as frequently as they wish.

In order to provide efficient information access in such environments, relevant data is often retrieved from several sources, integrated as necessary, and then materialized at a view site. Besides providing simplified and customized information access to users who may not have the time nor skill to identify and retrieve relevant information from all sources, materialized views may also offer more consistent availability shielding users from the fact that some of the underlying ISs may temporarily become disconnected as well as offering better query performance as all information can be retrieved from a single location.

However, materialized views in such evolving environments also introduce new challenges to the database community [Wid95]. We refer to any process that changes the view definition or the view extent (i.e., materialized view data) as *view adaptation* process. In Section 2 we propose a taxonomy of view adaptation problems in evolving environments based upon types of changes that either a view or an information source could undergo. This taxonomy represents a suitable classification based on which we can define as well as differentiate between different view adaptation problems, such as materialized view maintenance [GM95, Wid95], view redefinition [GMR95, MD96],

[1] These may include information such as their schemas, their query interfaces, as well as other services offered by the information sources.

etc.

We also use the taxonomy to identify a new view adaptation problem, called *view synchronization*, that corresponds to the process of view definition adaptation triggered by capability changes of ISs.

We propose a general framework, which we call the Evolvable View Environment (EVE), for this process of view adaptation triggered by capability changes of ISs. In EVE, views are specified using an SQL extension which enables view creators to explicitly state the evolution preferences, e.g., whether dropping or changing a view component is acceptable. Our algorithm for view synchronization then attempts to construct a modified view definition for a view V in response to a capability change of an IS that affects V, while meeting the evolution preferences.

The rest of the paper is organized as follows. Section 2 presents our general taxonomy of view adaptation problems, while Section 3 characterizes problem subspaces of specific view adaptation problems. Sections 4 and 5 characterize our solution for the view synchronization problem, namely, the EVE framework and its components. Section 6 discusses some related work and Section 7 concludes the paper.

2 A Taxonomy of View Adaptation Problems

In order to define our *view synchronization* problem, we now provide a general characterization of view adaptation problems, including both our new as well as other already well studied problems, such as view maintenance and view redefinition. The proposed taxonomy is based upon the types of changes to the view and base information, and the desired level of view adaptability in the context of changes. Further, for each view adaptation problem, we present the dimensions that characterize its subproblems.

2.1 Dimensions of View Adaptation Problems

We identify four dimensions that define the coordinates for different view adaptation problems (Fig. 1):

IS-Related Changes dimension: There are three types of changes that can be made in such environments at the information supply side, namely (1) changing the *metadata*, such as adding the fact that a relation R in IS_1 is contained in another relation S in IS_2, (2) changing the *capability* of an IS, such as deleting/adding an attribute or a relation, and (3) updating the (data) content of an IS, such as adding a new tuple to a relation.

IS Willingness to Preserve dimension: This dimension refers to the willingness of an IS to conform to the information needed by views defined on top of it. Possible values along this dimension correspond to the IS is willing and able to preserve *all, some*, or *not any* of its capabilities (either query interface, schema or even data content) referenced in a view V — denoted by *Yes*, *Some*, and *No* values on this dimension, respectively. If an IS for example opts to continue to preserve its data as it is in use by at least one of its derived views V, then V's data and/or definition would not need to be adapted after an IS change.

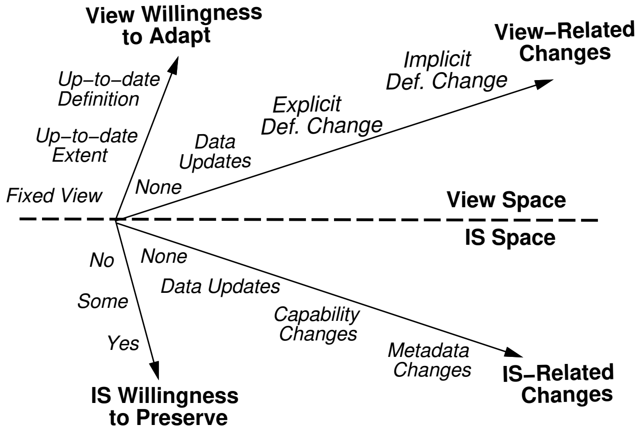

Figure 1: A Taxonomy of View Adaptation Problems.

View-Related Changes dimension: Most previous work in the materialized view literature has focussed on processing changes at the (view) content level, either triggered by updates at the information site or by view updates specified directly through the view interface (*Data Updates* value on this dimension). However, similarly to schema changes at an IS, capability changes also could take place for a view. Such definition changes could be either *explicitly requested* by the view maintainer, or they could be *implicitly triggered* due to other changes in the underlying environment.

View Willingness to Adapt dimension: This dimension considers the degree of desired flexibility in view adaptation in the context of changes, i.e., if the *view data* or *view definition* needs to be adapted after an IS-related or view-related change. This dimension refers to the criticality of different components of the view (e.g., view extent, view definition (VD), etc.) during the lifetime of the view, in order for the view to still be useful to the view user. One option (denoted by *Fixed View* value on this dimension) corresponds to not being able to handle any changes affecting the view - be it data or schema. The second option (denoted by *Up-to-date Extent* value on this dimension) corresponds to the view user's desire to maintain the *view extent* after any view-related or IS-related changes so that it always corresponds to the most up-to-date value set. The third option, which is the key focus of our work, corresponds to the view user's desire to adapt the *view definition* under ISs capability changes. This is denoted by *Up-to-date Definition* value on this dimension in Figure 1. The later represents a large range of options such as: (1) preserving the view interface and the original view extent while allowing the view definition to be changed, e.g., taking data from other sources, (2) preserving the view interface but accepting any view extent returned by the new view definition, (3) allowing some components to be dropped if no replacements can be found, versus (4) placing no constraints on the view adaptation process – which can survive any kind of capability changes.

2.2 Classification of View Adaptation Problems

In Figure 1, we depict the four dimensions that we use to define several view adaptation problems characterized by specific combinations of the coordinates. A set of four coordinates, one from each dimension, corresponds to a set of **apriori** conditions of the evolving view environment (on either IS or view space) that leads to a specific view adaptation process.

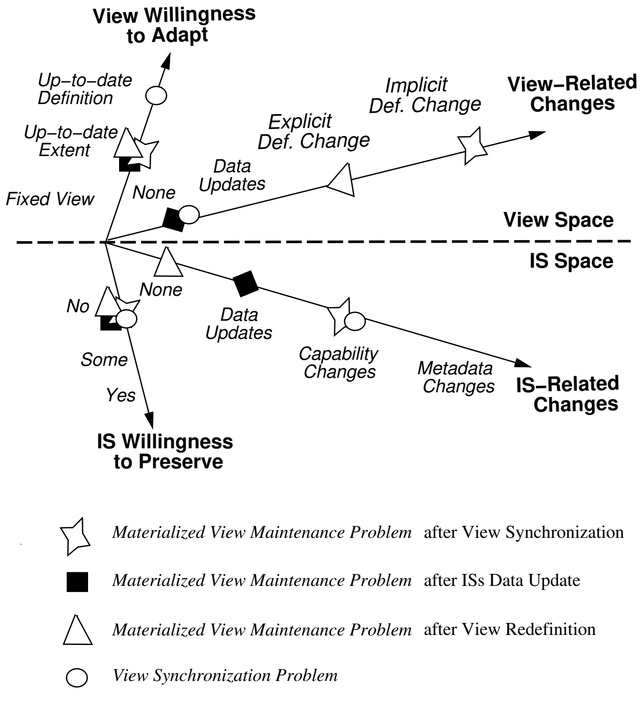

Figure 2: Problem Spaces in our View Adaptation Taxonomy.

1. *View Synchronization* is defined as a dynamic process that adapts the view definition triggered by IS-related capability changes. The coordinates that define this problem space are marked by circles in Figure 2. One key distinction of this problem as compared to others is that it is characterized by the *Up-to-date Definition* value on the **View Willingness to Adapt** dimension. This characterizes the fact that the view user desires for the *view definition* to adapt with changing IS capabilities instead of the view becoming undefined. The *None* value on the **View-Related Changes** dimension means that apriori no other change occurs on the view. The changes of the underlying ISs we consider are capability changes as indicated by the *Capability Changes* value on the **IS-Related Changes** dimension. No assumptions about the the modified ISs being willing to preserve their old capabilities are being made, as indicated by the *No* value on the **IS Willingness to Preserve** dimension[2].

2. The rest of the problem spaces depicted in Figure 2 are defined by a common view adaptability preference: maintaining the extent of the materialized view up-to-date as indicated on the **Willingness to Adapt** dimension.

   (a) The problem of *materialized view maintenance after base relation updates* is well studied in the database literature (see [GM95, Wid95] for an overview of problems and solutions). Its coordinates are marked by black squares in Figure 2. The problem is characterized by (1) the view has no explicit changes apriori, (2) the IS changed its data extent, (3) the view user wants the view extent up-to-date, and (4) the IS did not preserve its old data.

   (b) The problem of *materialized view maintenance after view redefinition* has been recently studied by Gupta et al. [GMR95] and by Mohania et al. [MD96]. Its coordinates are marked by triangles in Figure 2. The main difference to above is that the process is (1) triggered by the view's explicit change of its definition, and (2) *no* changes at the IS side occurred.

   (c) A new problem - *materialized view maintenance after view synchronization* - is now been given a formal framework by our taxonomy. In the context of view synchronization when the view definition is adapted to the IS capability changes, the view extent

[2] For this to be possible, we have to find a model to express evolution preferences for the desired view adaptability properties for different view elements under capability changes. See Section 4, for our solution to this issue.

also needs to be updated in response to view synchronization. This is what this new view maintenance problem is about. It is defined by the apriori conditions: (1) the view definition was synchronized, i.e., its definition was *implicitly changed* by the view synchronization process, (2) the IS changed its capability (this information is needed by the process of view maintenance in order to know what exactly was the change that triggers it), (3) the view user wants the view extent up-to-date, and (4) the IS did not preserve its old data nor schema.

The distinguishing characteristic of this maintenance problem over the other two is that potentially both ISs capabilities and data are changed (e.g., after the attribute A of relation R is deleted, relation R's data is changed as well, that is the column corresponding to attribute A is deleted from R, and hence both are not accessible for maintaining the view derived from it). This problem space brings new challenges as the changes in a view definition due to synchronization are in general complex as we discuss in Section 5. The coordinates of this problem are marked by stars in Figure 2.

3 Dimensions of Different View Adaptation Problem Spaces

For each view adaptation problem identified in Section 2.2, we describe next the dimensions that classify subproblems in each problem space. We emphasis in particular the *View Synchronization* problem as it is new and the target of our project work.

3.1 Problem Space of View Synchronization Problem

For the *View Synchronization* problem, we identify three dimensions along which the complexity of view synchronization algorithms can be characterized (Figure 3). Selections of design choices along these dimensions results in specific subproblems, each worth future investigation.

IS Capability Changes dimension: This corresponds to capability changes in databases such as *delete attribute*, *delete relation*, etc. Generally, these types of changes make the views derived from using these components undefined, leaving the view user with no choice but redefining the view completely. It is this problem that we address in this paper: what kind of extra information we must have at the view site in order to be able to redefine the view so that it survives after such changes.

Complexity of Meta Knowledge dimension: This corresponds to the description of capabilities and content of each IS plus of relationships between ISs. Such knowledge allows our system to reason about finding appropriate replacements of view components when evolving a view definition. One example of meta knowledge is that an identical duplicate of a relation R at IS_1 may also be available in another site IS_2, i.e., "$\text{IS}_1.R \equiv \text{IS}_2.S$". When the relation R is deleted from IS_1, the view synchronization process is able to preserve the affected view definitions that reference R by replacing $\text{IS}_1.R$ with $\text{IS}_2.S$. The more knowledge our system has about the information space, the higher chance it has to preserve the affected views when a capability change takes place at an IS - though of course the more complicated the algorithm for synchronization may become.

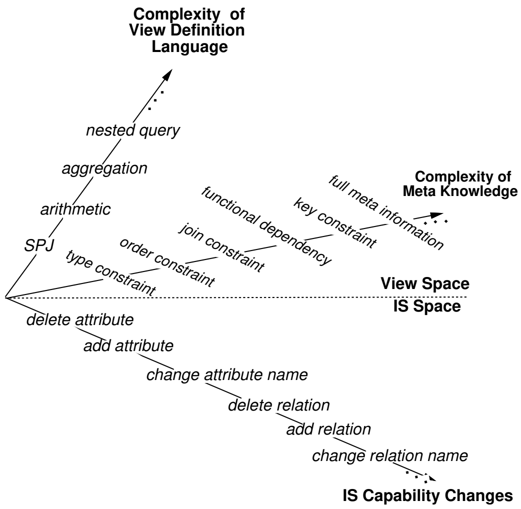

Figure 3: Problem Space of View Synchronization.

Complexity of View Definition Language dimension: This can be classified according to the view components allowed in the view definitions. The simplest view definition is specified by using the conventional SELECT-PROJECT-JOIN structure with WHERE clause containing conjunctive clauses only, denoted by *SPJ* in Figure 3. More complex views allow arithmetic, aggregation, negation, nested queries, and so on in their view definitions. If more complex language constructs are used as part of a view definition, then it is more likely that sophisticated algorithms are required to preserve the view interface (and the view extent) under capability changes.

In Section 5, we describe a view synchronization algorithm that addresses the points in this problem space of the following coordinates: (1) *delete attribute* value on the **IS Capability Changes** dimension; (2)

a selected subset of types of meta information on the **Complexity of Meta Knowledge** dimension; and (3) *SPJ* point on the **Complexity of View Definition Language** dimension.

3.2 Problem Space of Materialized View Maintenance After View Synchronization

After the view synchronization process, the view, if materialized, must be updated accordingly: this is what we coin the *materialized view maintenance after view synchronization* problem (Figure 2). We describe next the dimensions that characterize this new problem space depicted in Figure 4.

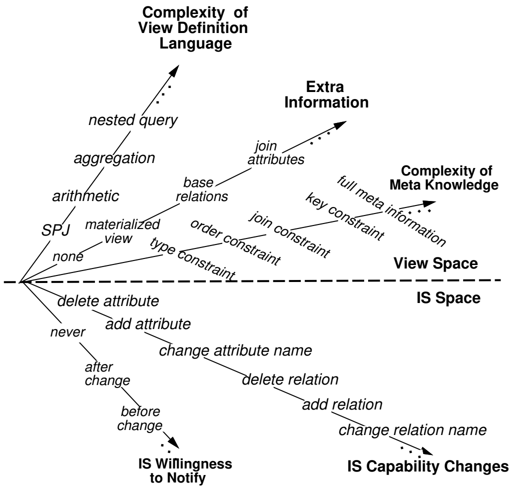

Figure 4: Problem Space of View Maintenance after View Synchronization.

Complexity of View Definition Language dimension: This characterizes the complexity of the view definition as done for view synchronization (Figure 3). **Extra Information** dimension: This dimension is concerned with the issue of what information besides the old and new view definitions is available for the purpose of facilitating view maintenance and possibly achieving view self-maintainability after view synchronization. The amount of information available for view maintenance can range from *fully duplicated* base relations, *partially duplicated* base relations – such as join attributes, the attributes in the SELECT clause, derive-counts, and auxiliary join tables, to no data from base relations available at the view site.
Complexity of Meta Knowledge dimension: See description in the view synchronization problem space.
IS Capability Changes dimension: See description in the view synchronization problem space above.
IS Willingness to Notify dimension: This dimension considers whether, and if so, when ISs notify the view site of their changes. Choices here are that the IS does *not notify*, *notifies only after*, or *notifies the view site even before* the change actually takes place. A more optimal re-organization may be able to take place at a view site if it gets informed of a potential change (especially a deletion) before a change takes effect. In that case, it would have the chance to copy the affected data over to the view site, to utilize meta knowledge about the to-be-deleted IS component to identify alternate sources for information supply, etc.

3.3 Other Problem Spaces

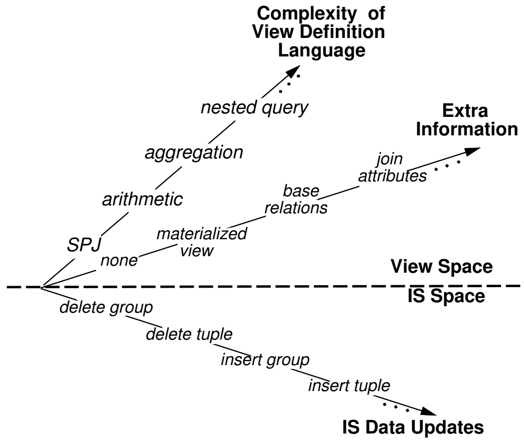

Figure 5: The Problem Space of View Maintenance after ISs Data Update.

Figures 5 and 6 depict the dimensions of the well known problem spaces: *Materialized View Maintenance* after base relation updates [GM95] and *Materialized View Maintenance* after view redefinition [MD96, GMR95].

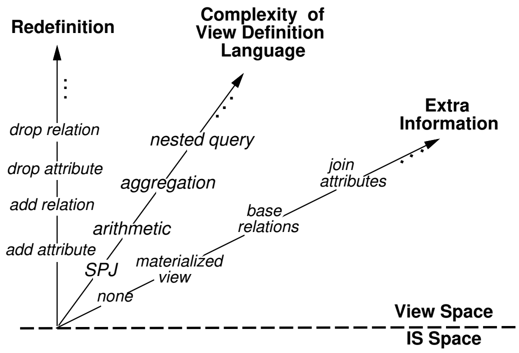

Figure 6: The Problem Space of View Maintenance after View Redefinition.

4 View Evolution in Evolvable View Environments (EVE)

The *view synchronization* problem tempts to evolve a view, if its view definition is affected by a capability change at a underlying IS, by finding appropriate replacements for the affected view components or by dropping non-essential view components if appropriate replacements cannot be found. We have designed the Evolvable View Environment (EVE) to tackle the problems concerning view synchronization and view maintenance after synchronization in dynamic environments (Figure 7). We give an architectural overview of the EVE framework next and outline our key design decisions.

IS Registration. Our environment can be divided into two spaces, i.e., the view space and information space. The information space is populated by a large number of external ISs. External ISs are heterogeneous, distributed, and autonomous; and they join, leave, or change their capabilities dynamically. An IS is "integrated" in the global framework via a mediator (i.e., the information source interface (ISI)) that serves as a bridge between the information space and the view space. The main functionality of an ISI is to translate the messages specified in the underlying data definition/manipulation languages into a common language used in the view site, and vice versa. The ISI may be intelligent so that it can extract not only raw data, but also meta information about the IS, such as changes at the schema level of the IS, performance data, or relationships with other ISs. Any IS that supports a query interface can participate in our environment.

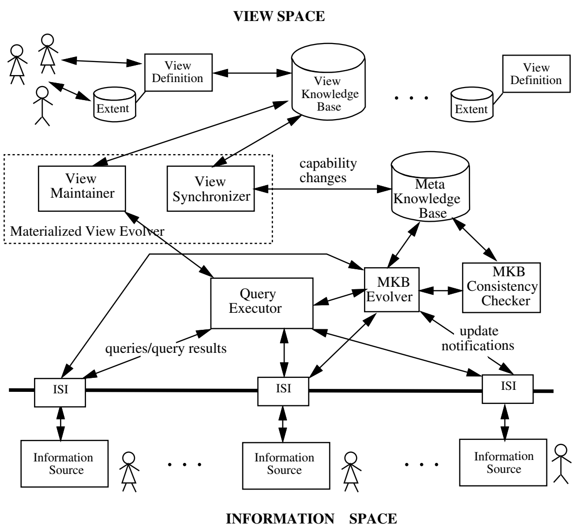

Figure 7: The Framework of Evolvable View Environment (EVE).

Meta Knowledge Base (MKB). When an IS joins EVE, it advertises to the MKB its capabilities, data model (e.g., the semantic mappings from its concepts to the concepts already in the MKB), and data content. The information providers have strong economic incentives to provide the meta knowledge of their individual ISs as well as the relationships with other ISs, since populating the MKB not only makes their data known by the view users, but it also increases the data utilization of their data set (especially, if they offer the same information at a better price).

We have designed a model for information source descriptions (MISD) [LNR97] that is capable of describing the content and capabilities of heterogeneous ISs, to be entered into the MKB. MISD captures meta knowledge such as an attribute must have a certain type (type integrity constraint), one relation can be meaningful joined with another relation if certain join constraints are satisfied (join constraint), a fragment of a relation is partially or completely contained in another fragment of some other relation (partial/complete information constraint), and so on [LNR97]. The IS descriptions collected in the MKB form an information pool that is critical in finding appropriate replacements for view components when view definitions become undefined (See Section 5) and for translating loosely-specified user requests into precise query plans [NR97].

Global Consistency Checking Across Sources. There are two types of inconsistencies (related to meta knowledge) in EVE. The first one is that constraints expressed in the MKB do not correspond to the information actually provided by ISs; and the second one is that different assertions in the MKB contradict with each other. The first type of inconsistency occurs when (1) either an IS provider makes an error when entering a MISD description, (2) an update occurred at one IS that causes a constraint that used to hold to become invalid, or (3) the usage and hence content of an IS changes over time without proper notification to the MKB. For example, the information provider for IS_1 inserts the fact that the relation R is equivalent to a relation S in another site IS_2 into the MKB. Now, the provider of IS_2, that is not aware of this assertion made about S in IS_2, inserts a new tuple t that makes the assertion become false.

There are alternative approaches for resolving this inconsistency. For example, (1) insert the tuple t into the relation R as well, (2) reject the insertion into S, (3) modify the invalid assertion in the MKB so to make it valid (i.e., in this case change "$IS_1.R \equiv IS_2.S$" into "$IS_1.R \subset IS_2.S$"), or (4) remove the invalid assertion from the MKB. Since checking and enforcing constraints across distributed autonomous ISs is an extremely difficult problem all on its own, in this work we assume that providers of individual ISs are in charge of assuring that their data is consistent with the meta knowledge collected in the MKB. We do not at this time incorporate a tool into our EVE framework that checks possible inconsistencies. However, once being notified about the entry or removal of some data item by an IS, EVE will notify the creators of all constraints in the MKB that may possibly be violated by this data modification. For example, on inserting a new tuple t into the relation S in the above example, both the providers of S and R are notified that the update occurred and that the constraint "$\mathrm{IS}_1.R \equiv \mathrm{IS}_2.S$" may now be inconsistent. It is up to the providers of IS_1 and IS_2 to determine how to handle this situation, once given the notification.

MKB Consistency. The second type of MKB consistency concerns conflicts between the constraints entered in the MKB, and thus can be detected by our MKB Consistency Checker module without help from the IS providers. One example of this type of conflict is that one information provider declares that a relation R of IS_1 is a strict subset of a relation S in another site IS_2, and at the same time the provider of S claims that the extent of S is a strict subset of of R. This is clearly an inconsistency. Our MKB consistency checker discovers such controversial meta knowledge using various types of inference techniques. Once detected, inconsistent assertions are reported to responsible information providers to have the differences resolved.

MKB Evolution. When an underlying IS makes a change to its capabilities (e.g., adds a new relation), the MKB no longer reveals the IS correctly in the sense that the meta knowledge describing the IS and the actual capabilities of the IS are distinct. For this, we have designed the MKB Evolution process to react to capability changes in the information space. In our framework, each IS will via the ISI interface notify the MKB of any such capability changes so that they can be properly registered in the MKB. The MKB Evolver module will then take appropriate actions to update the MKB [NLR97]. For example, deleting of an attribute A from a relation S may cause the MKB evolver to modify a subset constraint between two relations S and R, e.g., "$S \subset R$", into a simpler constraint "$S \subset$ (project all attributes of R besides A from R)". In other cases, some constraints may have to be completely removed from the MKB if they contain references to the deleted attribute.

Language for Information Access. For intelligent information access in such environments, we need a language that can express a user's demand for information both for a one-time (naive) requester as well as for constructing a materialized view site by a more seasoned user (a view developer). An adequate language must thus support query specifications ranging from a vague ontology-based information request [NR97] to a precisely elaborated query specification that details what piece of information is to be retrieved from which particular IS (as assumed in OEM by Papakonstantinou et al. [PGMW95]). We have designed the DIIM query language [NR97] so to be able to handle this wide range.

Query Planning. Our system incorporates algorithms for mediating between the requests for information (e.g., a view definition) and the actual information supply available in ISs as characterized in our MKB. In particular, we have developed the DIIM query planner that refines a loosely-specified user query into a precise query that meets all constraints expressed in the MKB and is executable within the environment [NR97]. Our proposed query semantics define a natural way of refining a vague query by rewriting it into a more restricted query that is consistent with both IS descriptions and the original query definition. A refined query uses join constraints to extract information from different ISs and at the same time is in agreement with their query interfaces. For this, we introduce the notion of *connected relations* as a natural extension of the concept of full disjunction [RSU95]. In the default case when only natural joins are defined in the IS descriptions in the MKB we have shown that the semantics of these two concepts (connected rules and full disjunction) are equivalent [NR97].

View Synchronization. We are applying query planning algorithms, such as the DIIM planner [NR97], to solve the view synchronization problem. (For more details see Section 5). We are continuing to investigate different reasoning techniques to determine progressively more complex means of preserving as much as possible the initial view [NLR97].

View Maintenance. The *view maintainer* tool in our EVE framework (Figure 7) is in charge of propagating data updates executed on an IS site following a capability change to the view site to bring the view content up-to-date after the view definition already had been changed by the view synchronizer.

5 Our Solution Approach to the View Synchronization Problem

In this section, we give an overview of the key constituents of our EVE framework, while a complete description of the EVE solution can be found in [LNR97].

5.1 Basics of the Evolvable-SQL View Definition Language (E-SQL)

A novel principle of our approach is to explore the evolution of views based on preferences specified by its creator, such as (1) whether the view component is essential for the view to be useful to the view user and (2) whether the view component is allowed to be obtained from ISs other than the IS originally stated in the view definition. The Evolvable-SQL view definition language (E-SQL) is our solution to this problem. E-SQL is an extension of SQL augmented with specifications for evolution. Essentially, E-SQL incorporates *evolving parameters* into SQL which allow the view definer to specify criteria based on which the view will be transparently evolved by the system under capability changes at the ISs.

We summarize the evolution parameters in Figure 8. Each row represents one type of evolution parameter in E-SQL. The four columns in Figure 8 represent the name, symbol, possible values and semantics, and default value for each evolving parameter, respectively. When the evolving parameter setting is omitted from the view definition, then the default value is assumed. This means that a conventional SQL query (without explicitly specified evolution preferences) has well-defined evolution semantics in EVE, i.e, anything the creator specified in the view definition must be preserved as originally specified in order for the view to be meaningful to the view user. The format of the E-SQL view definition language is given in Figure 9.

Next, we use one example to demonstrate the integrated usage of these evolution parameters, while a justification for the design of this language plus more examples can be found in [LNR97].

Example 1 *Let's assume a large web-based travel agency has a promotion for its customers who travel to Asia by air. The travel agency is either going to send promotion letters to these customers or call them by phone. Therefore, the travel agency needs to find the customers' names, addresses, and phone numbers. The query for getting the necessary information can be specified in SQL as follows:*

$$\begin{array}{rl} \mathsf{CREATE} & \mathsf{VIEW}\ \boldsymbol{Asia\text{-}Customer}\ \mathsf{AS} \\ \mathsf{SELECT} & \boldsymbol{C.Name,\ C.Address,\ C.PhoneNo} \\ \mathsf{FROM} & \boldsymbol{Customer\ C,\ FlightRes\ F} \\ \mathsf{WHERE} & \boldsymbol{(C.Name = F.Passenger)} \\ & \boldsymbol{AND\ (F.Destination = 'Asia')} \end{array} \tag{2}$$

Equation (2) is static, incapable of evolving, while IS capability changes are inevitable in a dynamic environment. We reformulate Equation (2) with view evolution parameters so that the view ***Asia-Customer*** *may survive in a changing environment. The resulting E-SQL view is shown in Equation (3).*

Assume the company is willing to put off the phone marketing strategy, if the customer's phone number cannot be obtained, e.g., the information provider of the ***Customer*** *relation decides to delete* ***PhoneNo****. This preference can be stated clearly in the* SELECT *clause of Equation (2) by the attribute dispensable parameter π. In addition, if the travel agent is willing to accept the customer information from other branches, we set the relation replaceable parameter θ in the* FROM *clause to true. Further, let's assume the travel agent is willing to offer its promotion to all the customers who travel by air, if identifying who travels to Asia is impossible (i.e., the second* WHERE *condition cannot be verified)* [3]*. This preference can be explicitly specified by associating the condition-dispensable parameter with that condition in the* WHERE *clause.*

$$\begin{array}{rl} \mathsf{CREATE} & \mathsf{VIEW}\ \boldsymbol{Asia\text{-}Customer}\ (\Delta = \supseteq)\ \mathsf{AS} \\ \mathsf{SELECT} & \boldsymbol{Name}\ (\varepsilon = false), \boldsymbol{Address}\ (\varepsilon = false), \\ & \underline{\boldsymbol{PhoneNo}\ (\pi = true,\ \varepsilon = true)} \\ \mathsf{FROM} & \boldsymbol{Customer\ C}\ (\theta = true), \boldsymbol{FlightRes\ F} \\ \mathsf{WHERE} & \boldsymbol{(C.Name = F.Passenger)}\ (\sigma = false) \\ & \textsc{and}\ \boldsymbol{(F.Destination = 'Asia')}\ (\sigma = true) \end{array} \tag{3}$$

5.2 Model of Information Source Description

While ISs may be based on disparate data models, when registered ISs translate their IS schemas into one common model supported by the EVE system (see [NR96, LNR97]). We now introduce the model of information source descriptions (MISD) (for a full description of MISD, see [LNR97]) we use to specify the meta knowledge currently collected in the MKB of EVE. We summarize the name (first column) and the syntax (second column) of the different types of constraints collected in the MKB in Figure 10.

In the following, we give some examples. The type integrity constraint:

$$Customer(Name) \subseteq String(Name)$$

says that the **Name** value of any customer is a string. The join constraint:

$$\mathcal{JC}_{Customer,FlightRes} = (Customer.Name = FlightRes.Passenger)$$

states that it is meaningful to combine the **Customer** and **FlightRes** relations, possibly taken from different ISs, as long as the customer's name is the same as the passenger's name. The partial/complete information constraint:

[3] 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.

View Evolution Parameter				
Parameter		Symbol	Semantics	Default
Attribute-	dispensable (AD)	π	*true:* the attribute is dispensable *false:* the attribute is indispensable	false
	replaceable (AR)	ε	*true:* the attribute is replaceable *false:* the attribute is nonreplaceable	false
Condition-	dispensable (CD)	σ	*true:* the condition is dispensable *false:* the condition is indispensable	false
	replaceable (CR)	β	*true:* the condition is replaceable *false:* the condition is nonreplaceable	false
Relation-	dispensable (RD)	α	*true:* the relation is dispensable *false:* the relation is indispensable	false
	replaceable (RR)	θ	*true:* the relation is replaceable *false:* the relation is nonreplaceable	false
View-	extent (VE)	Δ	don't care: no restriction on the new extent $\equiv$: 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	$\equiv$

Figure 8: View Evolution Parameters of the E-SQL Language.

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

where $\{B_1, \ldots, B_m\}$ corresponds to local names given to attributes preserved in V, $\{A_{s_{j,1}}, \ldots, A_{s_{j,i_j}}\}$ is a subset of the attributes of relation R_j with $j = 1, ..., n$; C_i with $i = 1, ..., k$, are WHERE conditions (primitive clauses) defined over the attributes of relations in the FROM clause. The parameters $\Delta, \pi, \epsilon, \alpha, \theta, \sigma$ and β are defined in Fig. 8.

Figure 9: Syntax of E-SQL Query.

$$\mathcal{PC}_{Customer,Branch} = (\pi_{Name,Address}(Customer) \supseteq \pi_{Name,Address}(Branch))$$

represents the fact that the customer information kept in the **Branch** relation is included in the customer information kept in the **Customer** relation from the headquarter IS.

5.3 View Synchronization Algorithm for Delete-Attribute Capability Change

When an attribute A of the **R** relation is deleted from IS1, our view synchronization process (VSP) searches the view knowledge base (VKB) for the affected views. For each of the affected views, VSP finds an acceptable view definition for the affected view, based on the evolution preferences specified in the E-SQL view definition, the type of capability change, and the meta knowledge in the MKB. In Figure 11, we present a flow chart of our view synchronization algorithm for the case when the capability change is *delete-attribute* (see [NLR97] for other view synchronization algorithms).

In this algorithm, VSP finds an appropriate substitute S.B for R.A, if (1) S.B has the same type as R.A, (2) it is meaningful to join S and R, i.e., a join constraint between R and S is found in the MKB, and (3) the new view extent that results after S.B replaces R.A in the view definition satisfies the view extent requirement.

Example 2 *Let's assume the information provider of the* ***Customer*** *relation decides to drop the* ***PhoneNo*** *attribute. When the VSP (view synchronization process) receives this delete-attribute capability change notification, it searches the VKB and finds that the* ***Asia-Customer*** *view (see Equation (3)) is affected. VSP searches in the MKB for an appropriate replacement for* ***Customer.PhoneNo*** *(Figure 11). If the phone number information cannot be found in the information space, then VSP drops the* ***PhoneNo*** *view component from the SELECT clause, registers the mod-*

Model for Information Source Description	
Name	Syntax
Type Integrity Constraint	$R(A_1, \ldots A_n) \subseteq Type_1(A_1), \ldots Type_n(A_n)$
Order Integrity Constraint	$R(A_1, \ldots, A_n) \subseteq \mathcal{C}(A_{i_1}, \ldots, A_{i_k})$
Join Constraint	$\mathcal{JC}_{R_1,R_2} = (C_1 AND \cdots C_l)$
Partial/Complete Constraint	$\mathcal{PC}_{R_1,R_2} = (\pi_{A_{i_s}}(\sigma_{\mathcal{C}(A_{j_1},\ldots,A_{j_l})} R_1) \theta \pi_{A_{n_s}}(\sigma_{\mathcal{C}(A_{m_1},\ldots,A_{m_t})} R_2))$

Figure 10: Model for Information Source Description.

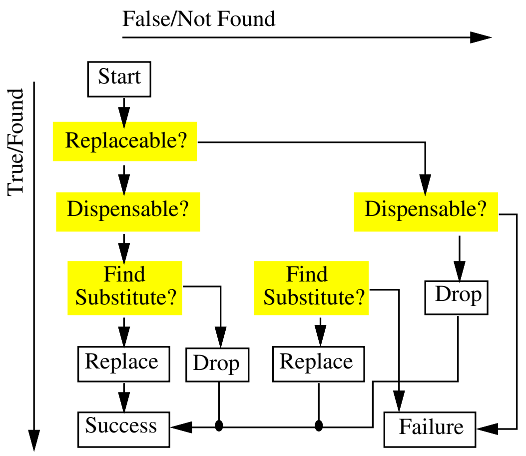

Figure 11: Flow Chart: View Synchronization Algorithm when Attribute Is Deleted from Select.

ified view definition with the VKB, and removes the ***PhoneNo*** *attribute from the original view extent.*

6 Related Work

To our knowledge, the view adaptation problems caused by capability changes in participating ISs (view synchronization and associated view maintenance problems) have not been studied before.

In the *MultiView* project, we have investigated the design and implementation of object-oriented view technology in a centralized setting [Run92, KR96a, KR96b]. We are also studying the application of this view technology to simulate schema changes instead of changing the base schema and thus affecting all users of the shared object-oriented database [RR95].

Gupta et al. [GJM96] and Mohania et al. [MD96] address the view redefinition problem. 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 EVE has determined an acceptable view redefinition. Their results are thus complimentary to our work.

Gupta et al. [GM95] identify the view maintenance problems according to four dimensions (e.g., the amount of information used for view maintenance) that constitute the problem space, but do not treat IS capability changes which is one of the key dimensions driving our taxonomic development.

In the work of Levy et al. [LSK95], external ISs are described relative to one unified world-view model. Levy et al. focus on choosing the right base relations for execution of queries expressed using world view relations, with the relationships between the world view relations and the relations of the underlying sources all expressed in that world view. The problem of view evolution, i.e., that the world view itself may evolve, is not handled in [LSK95].

Papakonstantinou et al. [PGMW95] pursue the goal of information gathering across multiple sources. Their proposed OEM language is traditional in the sense of being static, hence the view synchronization problem is not touched upon by their work.

In the University of Michigan Digital Library project [NR97], we have proposed the Dynamic Information Integration Model (DIIM) to allow ISs 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 IS exports when joining the DIIM system. We are in the process of adapting and incorporating the DIIM query planner as one thread of our view synchronization tool suite.

7 Conclusion

This paper is the first work to study the problem of view adaptation in dynamic environments. We bring forward a taxonomy of the view adaptation problems composed of four dimensions. We use our proposed taxonomy to classify and differentiate among different view adaptation problems, such as view redefinition and view maintenance [GM95, MD96, GMR95]. We also use it to define a new problem of view adaptation, which we call *view synchronization*, that corresponds to the process of adapting view definitions triggered by capability changes of ISs.

We propose the Evolvable View Environment

(EVE) architecture as a generic framework within which to solve view adaptation when underlying ISs change their capabilities. Design decisions and key components of the EVE solution, in particular, as they relate to the view synchronization problem, are also reviewed in this paper. In short, this paper has opened up a new direction of research by identifying view synchronization as an important and so far unexplored problem of current view technology in dynamic large-scale environments such as the WWW.

Acknowledgments. The authors would like to thank students at the University of Michigan Database Group and at the Database Systems Research Group at WPI for their interactions on this research. In particular, we thank Esther Dubin (CRA summer research student) and Xin Zhang for helping to proofread the paper and for implementing components of the EVE system.

References

[GJM96] A. Gupta, H.V. Jagadish, and I.S. Mumick. Data Integration using Self-Maintainable Views. *International Conference on Extending Database Technology (EDBT)*, 1996.

[GM95] A. Gupta and I.S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

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

[KR96a] H. A. Kuno and E. A. Rundensteiner. The *MultiView* OODB View System: Design and Implementation. In Harold Ossher and William Harrison, editors, *Theory and Practice of Object Systems (TAPOS), Special Issue on Subjectivity in Object-Oriented Systems.* John Wiley New York, 1996.

[KR96b] H. A. Kuno and E. A. Rundensteiner. Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views. In *IEEE International Conference on Data Engineering*, pages 310–317, 1996.

[LNR97] 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.

[NLR97] A. Nica, A.J. Lee, and E. A. Rundensteiner. View Synchronization with Complex Substitution Algorithms. Technical Report WPI-CS-TR-97-6, Worcester Polytechnic Institute, Dept. of Computer Science, July, 1997.

[NR96] A. Nica and E. A. Rundensteiner. The Dynamic Information Integration Model. Technical report, 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. *IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

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

[RSU95] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering Queries Using Templates With Binding Patterns. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 105–112, May 1995.

[Run92] E. A. Rundensteiner. *MultiView*: Methodology for Supporting Multiple Views in Object-Oriented Databases". In *18th VLDB Conference*, pages 187–198, 1992.

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