8

Extending View Technology for Complex Integration Tasks[1]

Matthew C. Jones and Elke A. Rundensteiner
Software Systems Research Laboratory
University of Michigan, Ann Arbor, MI 48109-2122
e-mail: mjones@eecs.umich.edu, rundenst@eecs.umich.edu

Abstract

In this paper, we present extensions to the MultiView object-oriented database view system that improve its ability to integrate electronic computer-aided design (ECAD) tools. Using an object-preserving algebra as the view definition language, the MultiView system supports data transformations to suit the needs of individual tools. However, an object-preserving algebra alone is not powerful enough to express recursive transformations such as transitive path derivations and the flattening of hierarchical structures. In order to provide these capabilities, we extend the view definition language with operators that permit complex transformations of the data. To achieve performance essential for ECAD tools, we introduce the mediator class as a general strategy for the materialization of these complex views. We compare the bounds on query and update performance of our set-based mediator class for the symmetric transitive closure with a more traditional object-based materialization strategy.

1 Introduction

Object-Oriented Databases (OODBs) are typically chosen to implement ECAD databases because they support the modelling of complex data [1,3,14,16]. Although integrated ECAD tools benefit from these database services, such as data integration and access control, these tools often employ different internal representations of design data. A schematic editor, for example, represents graphical entities hierarchically in the design schematic, including information about position and orientation. On the other hand, an analysis tool might represent the design as nodes and edges in a flattened graph. Even with current integration standards [4,6] the burden of transforming design data to and from the central database format falls upon the tool developer. This manual and often ad-hoc transformation process is labor-intensive and error prone. An integration system should provide a means for automating these data transformations. Additionally, if tools are to operate cooperatively and concurrently, tools are needed to simplify the communication of data changes between tools.

Recently, OODB *view systems* have been presented as a means for providing automatic reorganization of central data that has been customized according to a high-level restructuring request (i.e., a query). In this model, the tools in the system are *view users*, using central data that has been automatically organized by the view system. Our OODB view system, MultiView, [14] provides a *view definition language* (VDL) capable of restructuring data for each tool in the system. As such, it automates the transformation of data between the central format and the tool specific formats. In addition, the system provides the services necessary to maintain consistency between the central and derived data [11]. MultiView assures the correctness of data transformations and reduces undesired coupling between integrated tools. As a consequence, it increases the productivity of tool developers and integrators. However, the MultiView system is currently not capable of performing complex data transformations, such as the computation of closure, the traversal of paths, or the flattening of hierarchical graph structures.In this paper we present new operators for the VDL of the MultiView system. These operators permit it to fill the role of a central data store in a tool integration environment supporting complex data transformations. To provide the performance essential for ECAD applications, we introduce the *mediator class*, our abstraction for storing and maintaining the results of the transformation associated with a view operator. We address the design and performance of the mediator class associated with the computation of the symmetric transitive closure (STC) derivation.

In Section 2, we present view technology as the basis for tool integration. In Section 3, we discuss the MultiView system. In Section 4 we present the TC/STC operators and describe characteristics of the base and derived data. In Section 5 we explain the implementation of both an object-based and our mediator-based strategy for materializing the STC operation. We compare the performance of the mediator with the more general object-based materialization strategy in Section 6. We briefly describe other types of mediators in Section 7, and cover related work and conclusions in Sections 8 and 9.

---

1. This work was supported in part by NSF NYI #IRI-9457609, NSF RIA #IRI-9309076, Intel, and Digital Equipment Corporation.

F. J. Rammig et al. (eds.), *Electronic Design Automation Frameworks*

2 Applying Object-Oriented View Technology to Tool Integration

Object-oriented view mechanisms [1, 14] have been proposed as a means to provide customized data reorganization allowing access to centrally stored and managed data (*base data*) that is automatically restructured or *derived* into a preferred format. The view is considered *updatable* if changes to the derived data produce corresponding changes in the base data. Recently there has been considerable research into OODB view technology. However, with the exception of the O2 system [3] in alpha-test, commercial OODBS are not currently providing support for views.

Figure 1 illustrates the use of a view system as the basis for tool integration. A *data server* manages centrally stored data. A view system produces customized views in the form of *derived data*, generated by the view system for each tool (*view user*) in the environment. If the view system is distributed, each of the views is implemented by a client called the *view maintainer* that provides the view user access to the derived data and communicates changes between the data server and the view.

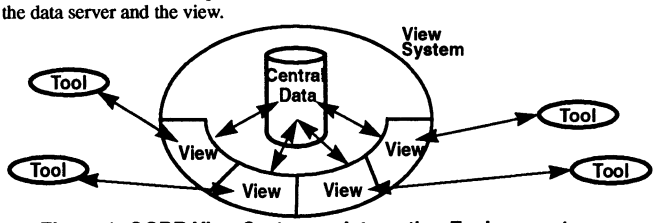

Figure 1: OODB View System as Integration Environment.

The view system may operate as a vehicle for the check-in and check-out of tool data - providing reorganization when tools read and write data. It may also act as a run-time integration system by propagating changes back to the base data (and subsequently to other active views) as changes are made to derived data.

The implementation of such a system requires solutions to *many* problems in data modelling, database views, and distributed systems. We identify key requirements for a view-based integration environment below, and describe where they fit within the scope of our work.

1. The tool integrator must be able to specify how data is to be reorganized. This *view definition* is usually specified using a view definition language (VDL). The VDL must be expressive enough to define complex views appropriate for ECAD applications. Our work extends the MultiView VDL accordingly.
2. The system must be able to create the view as defined by the tool integrator and provide the view user with access to the derived data. The access can be via a *materialized view* (implemented as actual view objects derived from base objects in the central database), or non-materialized view (implemented as an interface to base objects). Because performance is crucial in many ECAD applications, and because it is possible to express complex data transformations with the VDL, we present a solution that supports the materialization of derived data by the view maintainer.
3. If the view user makes changes to derived data, the system must propagate these changes to the base data (*view update*). The problem of *view consistency*, i.e., propagation of base data updates to views, only occurs when views are materialized. In such a case, the central data server must notify the view maintainer to make the appropriate changes to the derived data. We assume the existence of a notification mechanism, and accomplish view consistency by notifying the view maintainer through an update request which has been appropriately transformed for the associated view.
4. The tool integrator must be able to specify when and if updates are propagated to and from a view. This is accomplished using *update contracts*. Update contracts will be discussed in a future paper.

In this paper, we present an *extended query language* for defining complex view transformations, as well the development of a key abstraction for the materialization and maintenance of these complex views.

3 MultiView: The Object View System

At the University of Michigan, we have an on-going research project funded by NSF and Intel for the development of an object-oriented view support system, called MultiView [14]. MultiView has been successfully used to integrate ECAD tools in the domain of behavioral synthesis [15]. It is particularly suited to integration tasks such as hiding unneeded details from a tool, simplifying application specific processing by augmenting a view with customized functions, and precompiling information which is frequently needed by the tool.

For example, a simple placement tool may only be interested in adjacencies between components rather than the complete circuit netlist as stored in the central database. A view defined in MultiView can abstract away the component pins and explicit pin and net connections, and provide a derived attribute that models the adjacency

relationship. In this case, MultiView is hiding the unnecessary details of the pins, simplifying processing by encapsulating the adjacency attribute, and precompiling the adjacency information needed by the placement tool.

The MultiView system currently employs an *object-preserving* algebra as a query language for view definitions. Among its unique features, MultiView: (1) provides for full property inheritance for both base and virtual classes, (2) automates integration of virtual classes into a global schema, and (3) supports generation of complete view schemata rather than just virtual classes. The first prototype has been fully functional for over a year. It currently runs on a Sun SPARCstation and has been implemented on top of the Gemstone OODB [10].

To model the more complex restructuring employed in ECAD tools, such as flattening hierarchical graphs and deriving transitive relationships, we are now proposing extensions to MultiView. We have designed extensions to the query language to support more complex, recursive queries. In addition, we are designing extensions to the set-based data model to include sequences, graphs, and paths. Because of these added complexities we also must re-examine the implications to query processing and the view update and consistency problems.

In this paper, we enable the derivation of views optimized for ECAD applications by adding complex, object-creating operators. This optimization is twofold: (1) transformations on ECAD data are compact and easy to specify, reducing effort during integration, and (2) efficient access to the transformed data, providing the performance that is critical in ECAD applications.

4 Extending the VDL for Closure Computation

4.1 Characteristics of the Base Data

We now present the language extensions by which these complex views can be specified. We also characterize the kinds of derived views possible with these new operators.

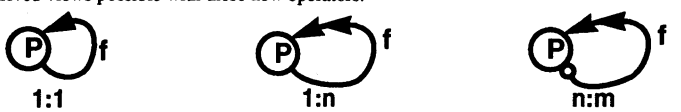

Figure 2: Recursive Schema Graphs.

Consider views derived from the base schemata illustrated in Figure 2. These schemata contain a single class `P`, and a relation **F** (modeled by the instance variable `f`) that relates two instances in `P`. We define how the instance variable `f` models the relationship **F** formally as:

$$\text{for } p_1, p_2 \text{ in } \texttt{P}\text{: } p_2 \text{ in } p_1.\texttt{f} \text{ iff } p_1 \,\mathbf{F}\, p_2 \tag{1}$$

The cardinality of this relation determines the properties of instance graphs possible with this schema. For example, an instance of the schema in Figure 2.a would result in a set of sequences of instances of `P`. Because these instance graphs have known properties, we are able to select our materialization strategy based upon this knowledge. Independently of whether or not the relationship **F** is transitive, the view user may wish to derive new attributes based upon the transitive closure of the attribute `f`. We consider two possible transitive derivations over the attribute `f`: the *symmetric* and *asymmetric transitive closure* known as STC_f and TC_f respectively.

4.2 Characteristics of the Derived Data

The derivation of TC/STC creates a derived attribute with more general semantics than the base attribute. For example, if the attribute `f` represents the relation **F** (Fanout) between two instances in `P` (`Parts`), then the attribute `c` derived from the TC on `f` represents the more general semantics of **C** (Fanout-Cone). The transitive closure on the specialized relation **F** is used to derive the more general relation **C**. In this example, Fanout is a specialization of the Fanout-Cone relation. We derive the more general TC_f relation as follows:

$$\text{for } p_1, p_2 \text{ in } \texttt{P}\text{: } p_1 \,\mathbf{F}\, p_2 \text{ implies } p_1 \,\mathbf{C}\, p_2 \tag{2}$$

$$\text{for } p_1, p_2, p_3 \text{ in } \texttt{P}\text{: } p_1 \,\mathbf{C}\, p_2 \text{ and } p_2 \,\mathbf{C}\, p_3 \text{ implies } p_1 \,\mathbf{C}\, p_3 \tag{3}$$

Figure 3 illustrates an example of a schema, a possible object graph, and the derived TC_f instance graph. Both equations (2) and (3) are applied to the base instance graph to produce the derived instance graph. The application of equation (2) represents attributes *directly derived* from the base data. Equation (3) produces the attributes indirectly, or *transitively derived*, from base data. Note that the instance variables shown in the schema are represented by edges in the instance graph. The instances of the class `P` are nodes in both the base and the derived graph. This means that changes to the instance variables `f` in the base objects correspond to edge deletions and insertions in the base and derived instance graphs.

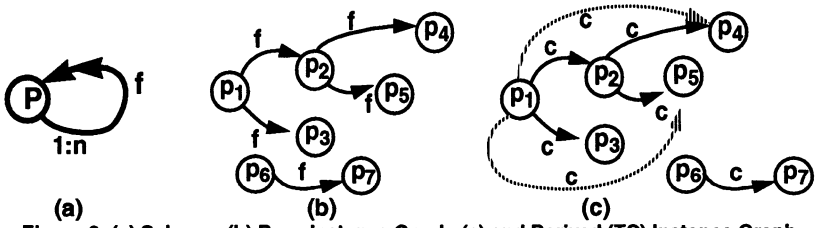

Figure 3: (a) Schema, (b) Base Instance Graph, (c) and Derived (TC) Instance Graph.

Note that a similar generalization occurs with the derivation of the STC. Here we derive a new relation **B** (InBlock) that relates all parts in a common combinational circuit block. To derive the STC, the relation **B** is directly derived from **F** by applying the derivation in (2), then it is *symmetrically derived* by applying:

$$\text{for } p_1, p_2 \text{ in } \mathbf{P}\text{: } p_1 \mathbf{B} p_2 \text{ implies } p_2 \mathbf{B} p_1. \tag{4}$$

and then transitively derived using formula (3). The resulting derived STC, a set of fully connected graphs for the base instance graph of Figure 3b, is shown in Figure 4. The derivation for an instance graph of N objects where $N \cong np$ divides the graph into n disjoint partitions connected by the derived attributes. The average size of each of these partitions is p.[1] If the base instance graph is connected by the base relation, then we have $p=N$ and $n=1$.

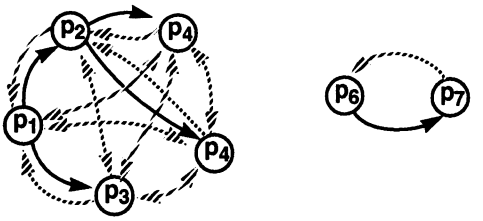

Figure 4: STC_f Instance Graph Derived from Figure 3.b.

4.3 Updates to/from Derived Relationships

The attributes such as those derived by equations (2), (3), and (4) are generalizations of the base attributes from which they are derived. The newly derived attributes represent a loss of information, and thus are called *lossy*. This has important implications for the derived relations and their use in a view system. The most important implication is that it is impossible for a view maintainer to incrementally update derived information that has "lost" information which contributes to the data derivation. We nullify the effects of the lossy derivation by recording enough information in the materialized view to distinguish between directly and indirectly derived attributes. This ensures that updates can be propagated, and that the impact on the view of base updates can be autonomously computed by the view maintainer. In general, it is not always possible to propagate updates made by the view user to indirectly derived attributes. This problem is addressed in more detail in Section 6.4.

5 Efficiently Materializing Closure Views with the Mediator Class

To achieve performance critical in ECAD, we employ view materialization. This materialization is invisible to the user of the view, however, the choice of representation for the materialized view can have a dramatic effect on the efficiency of queries and updates to the derived data.

Using the more traditional view materialization approach as utilized by relational systems, the materialization of a new relation is accomplished by creating a new object (tuple) representing a pairwise relationship between two objects. The result is a new class $\mathbf{T_{P,B}}$ whose instances contain two attributes p_a and p_b whose domain is **P**. An instance is placed into the extent of class $\mathbf{T_{P,B}}$ for each derived relationship. For this object-based materialization representing the derivation of the new relation **B**, we define the extent of $\mathbf{T_{P,B}}$ as follows:

$$\text{for } p_a, p_b \text{ in } \mathbf{P}\text{: object}(a{=}p_a, b{=}p_b) \text{ in } \mathbf{T_{P,B}} \text{ iff } p_a \mathbf{B} p_b. \tag{5}$$

1. For our analysis in the remainder of the paper, we assume that all subgraphs in the view have the same size p.

Both the size and time required to materialize this newly derived class are $O(n \times p^2)$. This bound comes from the fact that for each instance subgraph in the derived instance graph there are potentially p^2 newly derived relations. Recall that number of these subgraphs in the derived instance graph is n. The appeal of this object-based approach is its generality and simplicity. It can be used for any kind of derived relation. The data manager can manipulate this form of materialized view easily because it is a primitive form. We have found that one of its significant drawbacks is its inefficiency (both in space and time).

We thus introduce the *mediator class* as an alternative means to materialize the newly derived relation in the view. A mediator class is an optimized representation of the derived relation that accurately models the relation, and is able to propagate updates from the base data to the derived data efficiently. The mediator class is selected based upon known properties of the base data, and known properties of the derived relationship. The creation and materialization of a mediator class requires the following steps:

1. Identification of the appropriate mediator class for the current view of the base and derived data.
2. Materialization of the newly derived relation in mediator class objects.
3. An interface to the mediator to make it "look and feel" like the expected relationship semantics. These include access functions to query derived instance variables and methods to update them.

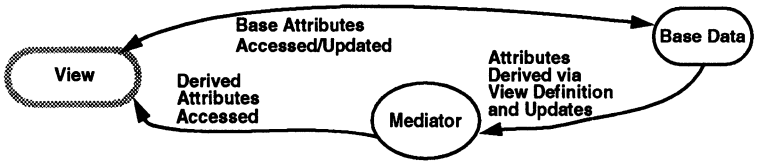

Figure 5: Mediator Class for Complex View Definition.

We assume that the view system knows enough about the characteristics of the base data to select a mediator class for the available query operators. Steps 2 and 3 are the focus of the sections that follow. In our mediator-based solution to materialization, we extend the derived class with a derived attribute that provides access to the mediator. The view system provides query and update operations on the derived attribute so that it has the appearance of a genuine attribute.

6 Implementation of Mediator Classes for Complex Views

This section discusses the implementation of a mediator class for the materialization of the STC. We analyze the performance of the implementation, and compare it with a simple object-based materialization approach.

6.1 Symmetric Transitive Closure

A user may wish to define a view comprised of the symmetric, transitive closure operation of an attribute. In an ECAD application, this type of transformation is typically used to combine a graph into a single entity, i.e., to gather segments in a schematic into a single net. We refer to the original schemata as illustrated in Figures 2.b and 2.c, and to the relations and attributes presented in Section 4.2. Our mediator implementation for the STC_f is independent of the cardinality of the attribute **f**, so we present the general solution here. Optimizations specific to a certain cardinality are discussed when they are significant.

We consider the user request to extend the class **P** with a derived attribute **b**, resulting in a virtual class **P'**.

```
P' = refine [ b = STCf(self) ] for (P).
```

Because of the nature of a symmetric, transitive relationship, the view system materializes the derived attribute **b** via a set-based mediator. The derived attribute **b** can be materialized using a set because of similarities between the properties of sets and the definition of symmetric, transitive closure. As expressed in the query, the new attribute **b** will provide access to the derived STC_f(**P**). **Same-set** membership in this set can then be used to model the appropriate transitive and symmetric relationships because of the following:

$$\text{Symmetry: } p_1 \text{ in } \textbf{same-set} \text{ as } p_2 \text{ implies } p_2 \text{ in } \textbf{same-set} \text{ as } p_1 \quad (6)$$

$$\text{Transitivity: } p_1 \text{ in } \textbf{same-set} \text{ as } p_2 \text{ and } p_2 \text{ in } \textbf{same-set} \text{ as } p_3 \text{ implies } p_1 \text{ in } \textbf{same-set} \text{ as } p_3 \quad (7)$$

Because same-set membership is a transitive and symmetric relationship, we define STC_f(**P**) to be of a set type, and materialize the resulting sets as instances of the mediator class. We also provide an interface to the mediator class to query and propagate updates to and from the view efficiently. Each object in **P'** has associated with it an instance of the mediator class. This mediator class is made available through the attribute **b** that is added with the view definition. Figure 6 shows the base instances with the derived attribute **b** associating each instances with the

mediator class instances used to represent the symmetric transitive closure relation. The mediator accurately represents the derived relationship by satisfying the following constraint:

$$\text{for } p_1, p_2 \text{ in P: } p_1 \text{ in } p_2.\mathbf{b} \quad \text{iff} \quad p_1 \text{ in } \mathrm{STC}_f(p_2) \tag{8}$$

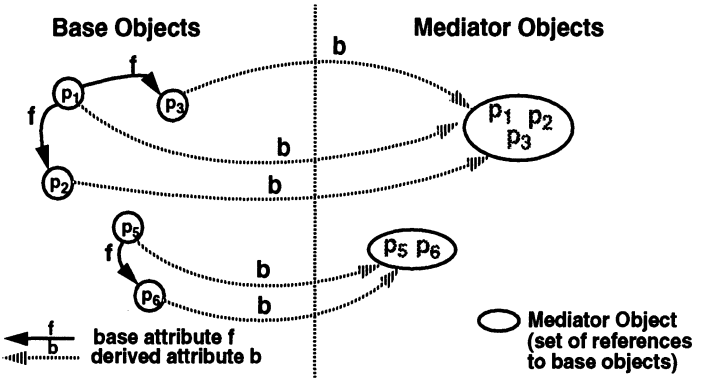

Figure 6: Mediators and the Derived Attribute b for $\mathrm{STC}_f(P)$.

6.2 Creation of the Derived Attribute and Mediator Class

The traditional object-based materialization approach derives objects to represent the STC. These objects are materialized in the class $\mathbf{T}_{P,B}$ using the following operations:

```
create_class T_P,B subclassOf Object with (a:P, b:P);
insert (select a:p1.self, b:p2.self from p1, p2 in P
        where p1 in STC_f(p2)) into T_P,B;
```

We create our mediator class with the following queries:

```
create_class M subclassOf STCMediator;
insert (select distinct the STC_f(p) from p in P) into M;
```

In both cases, a new attribute is created for the class **P'** which provides the interface to the derived and materialized STC relation. For the mediator case, the creation of the **b** attribute for the class **P'** is as follows:

```
P' = refine [ b = (select m from M where p in m)] for (p in P);
```

This defines the derived attribute to provide access to the mediator object that maintains the STC relation. For the object-based materialization the new attribute provides the interface to the query and update operations needed to access the STC materialized in $\mathbf{T}_{P,B}$.

6.3 Querying

We compare here the relative costs of performing queries on the traditional object-based materialization of the STC and our mediator-based materialization.

6.3.1 Membership Test in the $\mathrm{STC}_f(P)$

We next show how the two representations answer the query: `Is y in the STC_f(x)?`

Object-Based Representation: This tests for existence of an object that reflects the relationship between x and y.

```
return not empty(select * from T_P,B where a=x and b=y);
```

Because the size of the extent of $\mathbf{T}_{P,B}$ is $O(n \times p^2)$, this operation can be performed in time $O(\log(n \times p^2))$. This assumes the existence of proper indices to facilitate the search. The assumption of these indices impacts the estimated cost of insertions and updates, as well.

Mediator-Based Representation: The mediator determines if y is in the $\mathrm{STC}_f(x)$ with the $O(1)$ comparison of the reference attributes to the STC sets to see if they refer to the same set object.

```
return (x.s==y.s);
```

If both x and y have the same mediator set, then they are by definition in the STC of each other.

6.3.2 Retrieval of the $\mathrm{STC}_f(P)$ Relationship

We now discuss how the two representations answer the query: `What is the STC_f(x)?`

Object-Based Representation: In this case, we need to locate all objects in $\mathbf{T_{P,B}}$ that contain **x**.

```
return select b from T_P,B where a=x;
```

Although the size of the extent of $\mathbf{T_{P,B}}$ is $O(n \times p^2)$, and this operation takes $O(\log(n \times p^2))$ time, the actual performance may be bounded by the size of the STC which is returned. Since the number of objects which represents the STC is $O(p^2)$, this may become the bound on the performance of this query if p is sufficiently large. Because of the representation inherent in the object-based solution, it is not possible to return a reference to the set of objects in better than $O(p^2)$

Mediator-Based Representation: The STC is computed by returning the mediator set associated with x. Because the set is already grouped into a single object, a reference to the set can be returned in constant time. However, because the size of this set is $O(p)$, that is the bound on the performance of the query.

6.4 View Updates to Complex Derived Attributes

In general, the derived attribute **b** has ambiguous update semantics, as shown in Figure 7. This figure shows objects p_1 through p_5 in the base data related by the attribute **f**. The $STC_f(P)$ derives the fully-connected instance graph with the derived attribute **b**. Although neither materialization strategy uses an explicit graph to represent the derived data, it is useful for this example. The ambiguity in the view update occurs when the view user attempts to change the value of the derived attribute indicated with the α. If this reference to p_4 is removed from the attribute p_1.**b**, it is not possible to determine how to propagate this change to the base data. A single change to an indirectly derived attribute has many interpretations involving changes to base attributes. It might be possible for the view system to assess the minimal impact update necessary to propagate, but such a computation is expensive and of limited use. Merely changing the derived attribute value without propagating the change to the base is not an option, because that means the view no longer represents the $STC_f(P)$. After such a change it would no longer be possible to use the STC mediator, because the constraint on which the contents of the mediator is based is violated. For this reason, updates to the indirectly derived attributes are not permitted for the STC.

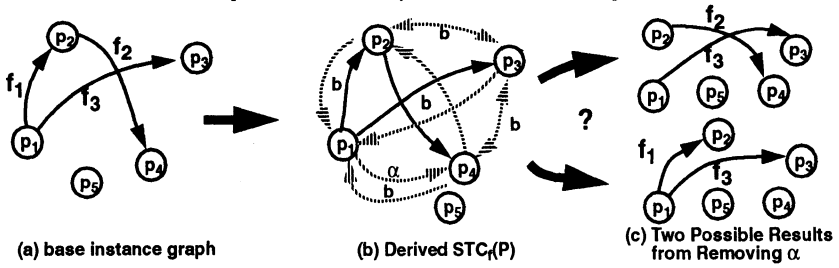

Figure 7: Ambiguous Update to Derived Attribute (deleting α).

6.5 View Consistency

If several tools use the base data simultaneously, then there must be a means to propagate updates performed on the base data to the views. We describe two updates which have direct impact on the derived view.

6.5.1 Reference Removal

We consider the effect of removing a reference to object y from the base attribute **f** of object x. This corresponds to an *edge deletion* in the base instance graph. The removal of the edge in the base instance graph may partition the graph, resulting in a corresponding partitioning of the sets materialized in the STC mediator.

Because the object-based approach is lossy, the view maintainer cannot *autonomously* determine the effect of such an update. As a consequence, the object-based approach requires that the central data manager communicates the resulting partitions to the view maintainer. The mediator class distinguishes between base and derived attributes, and consequently has enough information to autonomously compute the effect of the update on the view. The base data manager just reports attribute changes to the mediator. The mediator employs the same algorithm that would be used by the base data manager to determine the repartitioning of the base instance graph.

The partitioning algorithm builds two spanning trees of the base instance graph starting from both x and y. If the spanning trees are disjoint, then the attribute change partitions the graph, and the results of the partitioning

must be propagated to the derived data. This check requires $O(p)$ time to complete. If the attribute does not partition the base instance graph, no further action is required (the STC remains unchanged).

Object-Based Representation: The view maintainer receives update information from the base data manager in the form of two sets S_j and S_k, representing the partition of the instance subgraph produced by the removal of the reference to y. The following method updates the class $\mathbf{T_{P,B}}$ so that it reflects the removal of the reference to x.

```
foreach v in Sj
  foreach w in Sk
      remove object(a=v,b=w), object(a=w,b=v) from TP,B
```

The nested outer loops require $O(p^2)$ time, and the inner operation requires $O(\log(n \times p^2))$ time, for a total complexity of $O(\log(n \times p^2) \times p^2)$.

Mediator-Based Representation: After determining if the change to $x.\mathbf{f}$ partitions the base graph (and as a consequence computing the partitions S_j and S_k), the mediator updates its derived data with the following algorithm:

```
delete STCMediator(x.b);              // remove old set
new STCMediator(Sj);                  // create new sets
new STCMediator(Sk);
foreach v in Sj
  v.b = STCMediator(Sj);
foreach w in Sk
  w.b = STCMediator(Sk);
```

This algorithm associates each object with the new set that represents the STC of the object. It requires $O(p)$ time. In the case that the cardinality of the base attribute **f** is 1:n, we have a trivial partitioning of the base into the single element y, and the set consisting of `x.b`-y.

6.5.2 Reference Insertion

We consider the effect of inserting a reference to object y into the base attribute **f** of object x. This corresponds to an *edge insertion* in the base instance graph. The insertion of the edge in the base instance graph may result in the merger of two subgraphs in the STC derivation. Unlike edge deletion, however, the data manager takes no special actions to notify the maintainer of the STC view. It merely notifies the view maintainer about the addition to the attribute.

Object-Based Representation: The object-based representation receives update information in the form of an addition to the attribute value for the object x. The following method updates the class $\mathbf{T_{P,B}}$ so that it reflects the new value for attribute **f** of object x. It accomplishes the merger of the two subgraphs by creating an object associating every element of the subgraph in which x is a member, to every element of the subgraph in which y occurs.

```
foreach w in { select b from TP,B where a=x }
  foreach z in { select b from TP,B where a=y }
      insert object(a=w,b=z) into TP,B
      insert object(a=z,b=w) into TP,B
```

The outer loops have complexity $O(p2)$, and the inner operation requires $O(\log(n \times p2))$ time, for a total complexity of $O(\log(n \times p^2) \times p^2)$. The insertion time in the inner loop could be constant, but that would be at the expense of increased cost to other operations. The logarithmic insertion time is consistent with the overhead of maintaining an efficient structure which supports fast queries and deletions.

Mediator-Based Representation: The following algorithm updates the mediator:

```
newset = new STCMediator(x.b union y.b);
delete STCMediator(x.b);
delete STCMediator(y.b);
foreach v in (newset)        // link elements to new mediator
  v.b = newset;
```

This solution simply associates each affected object with the newly merged subgraph which represents the STC of the object. It requires $O(p)$ for the algorithm.

The table in Figure 8 shows clearly that the mediator-based materialization strategy achieves advantages in both time and space performance over the traditional relational materialization approach. The mediator also demonstrates efficient membership and retrieval performance, a property essential for ECAD analysis tools that do many more reads than updates.

7 Other Mediators

We have discussed the implementation of the mediator for the STC operator in the Section 6. In this section, we briefly discuss the TC mediators. Mediators for the derived TC differ from the set-based STC mediator, since the symmetry modeled by the set-based mediator is not appropriate when materializing the TC relationship.

Metric	Mediator-Based	Object-Based
Space Required	O(n x p)	O(n x p^2)
Creation Time	O(n x p)	O(n x p^2)
Membership Test	O(1)	O(log (n x p^2))
STC Retrieval	O(1)	O(log (n x p^2))
Edge Deletion	O(p)	O(log (n x p^2) x p^2)
Edge Insertion	O(p)	O(log (n x p^2) x p^2)

Figure 8: Comparison of Mediator-Based and Object-Based Materialization Strategies

7.1 TC Mediator for 1:1

We refer to the schema illustrated in Figure 2.a, and consider the view definition:

```
P' = refine [ c = TCf(self) ] for (P).
```

If the cardinality of the attribute `f` is 1:1, we can materialize and maintain the TC_f with a mediator class containing ordered sets (or sequences). For each element *e* in the sequence we define a sequence number seqnum(e), that identifies the position of each element *e* in the ordered set. We then model the TC_f implicitly using the same-set test along with the seqnum(e) associated with each element *e*. We state the definition of the TC_f in terms of operations on the mediator as:

$$\text{for } p_1,p_2 \text{ in P: } p_2 \text{ in } TC_f(p_1) \text{ iff } p_2 \text{ in same-set as } p_1 \text{ and seqnum}(p_2) > \text{seqnum}(p_1) \quad (9)$$

An object-based materialization of the TC_f performs similarly to the object-based materialization of the STC_f so we will not discuss the performance of the object-based materialization in this section.

The mediator provides a membership test (see Section 6.3.1) on the materialization of the TC_f by comparing references to the associated ordered set and comparing sequence numbers. This can be accomplished in $O(1)$ time. The $TC_f(p_n)$ can be retrieved for an object p_n in constant time, provided that the mediator is capable of returning a reference to a subsequence in the ordered set, otherwise the operation requires $O(p)$ time to build the sequence to return.

Updates on the base data can be efficiently propagated to the mediator. If an edge in the base instance graph is removed, this results in a partitioning of the ordered set materialized in the mediator. The edge itself provides the point in the sequence where the cut must be made. The result is two sequences representing the two new partitions in the base instance graph. This operation can be accomplished in $O(p)$ time. If an edge in the base instance graph is added, this results in the merger of two base instance graphs. The mediator merges the corresponding ordered sets by concatenating them. The operation requires $O(p)$ time.

We materialize the TC for instance graphs containing loops by using a set representation similar to the STC mediator objects. Because all objects in a loop are transitively related to each other, the set models this relationship accurately. Updates that break loops eliminate the associated set-based mediator and create a corresponding ordered-set mediator. Similarly, the mediator handles updates which introduce loops (i.e., joining the end of a sequence to the beginning) by creating the corresponding set-mediator and removing the ordered set mediator. These operations require $O(m)$ time.

7.2 TC_f for Other Cardinalities

The precise materialization strategy for TC_f of other cardinalities of `f` is still an open question. We are currently investigating strategies for materialization which balance performance for both queries and updates. We can improve the performance of queries by utilizing topological ordering in much the same way that we employ sequence numbers for the 1:1 case.

8 Related Work

The Gandalf system [8] was one of the earliest environments to use views as the basis for tool integration. This system used "display-only" views to provide the data for various tools in a software development environment. Updatable views were proposed in this work, but not implemented. In the FICOM system [2], a data manager is

presented that supports incremental consistency maintenance between different levels of abstraction in a CAD data manager. However, the representations are limited to those already implemented in the data manager. Furthermore, the FICOM system does not support materialization within the integrated tools.

In [9], the authors present the super-key class as a mechanism for enabling updates on materialized paths in object databases. Although the model they present is general, no implementation is discussed.

9 Conclusions and Future Work

In this paper, we have discussed extensions to the view definition language of MultiView to provide complex transformations as required by ECAD tools. We have proposed the addition of a mediator class to provide for the efficient materialization of certain classes of complex views. In addition, we presented the development of efficient materialization strategies which include support for updatable views. These contributions provide essential foundational technology for an integration support tool in an ECAD framework.

We plan to continue the development of update semantics for these complex structures, as well as classify the types of transformations which are appropriate for the integration of ECAD tools. This includes the evaluation of materialization strategies for the transitive closure operation, and other path transformations. We are also developing the use of our mediator class as a super-key class for more general path materialization.

Acknowledgments. Matthew would like to thank Karem Sakallah for financial support while he was "finding this topic". Many thanks go to Harumi Kuno and Tricia Jones for reviewing an earlier draft of this work.

10 References

[1] S. Abiteboul and A. Bonner, "Objects and Views," in *Proc. of the ACM SIGMOD 91*, 1991, pp. 238-247.

[2] R. Armstrong and J. Allen, "FICOM: A Framework for Incremental Consistency Maintenance in Multi-Representation, Structural VLSI Databases," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1992, pp. 336-343.

[3] O2 Technology. *O2 Views User Manual*, version 1 edition, December 1993.

[4] CFI-DR-TSC, "Design Representation Electrical Connectivity Information Model and Programming Interface," *CFI Publication*, Version 1.0.0, 1992

[5] CFI-ITC-TSC, "Inter-Tool Communication Architecture," *CFI Publication*, Version 1.0.0, 1992

[6] L. Claesen, R. Severyns, P. Six, W. D. Rammelaere, H. D. Man, J. Cockx, P. Reynaert, and G. Shrooten, "Open Framework of Interactive and Communicating CAD Tools," in *Tool Integration and Design Environments*, F. J. Rammig, ed., North-Holland, 1987.

[7] M. Jones, and E. A. Rundensteiner, "Mediator Classes for the Efficient Materialization and Update of Complex Views", Electrical Engineering and Computer Science Dept., University of Michigan, Ann Arbor, Tech. Rep., In Preparation.

[8] D. Garlan, "Views for Tools in Integrated Environments," in *Advanced Programming Environments*, Springer-Verlag, 1986, pp. 314-343.

[9] S. Konomi, T. Furukawa, and Y. Kambayashi, "Super-Key Classes for Updating Materialized Derived Classes in Object Bases," in *Proc. DOOD Conference*, Dec. 1993.

[10] H. A. Kuno and E. A. Rundensteiner, "Developing an Object-Oriented View Management System," *IBM CASCON*, Oct. 1993, pp. 548 - 562

[11] H. A. Kuno and E. A. Rundensteiner, "Materialized Object-Oriented Views in MultiView," in *Proc. Fifth International Workshop on Research Issues on Data Engineering: Distributed Object Management (RIDE-DOM '95)*, March 1995.

[12] J. Miller, K. Gröning, G. Schulz, and C. White, "The Object-Oriented Integration Methodology of the CADlab Work Station Design Environment," in *Proc. IEEE/ACM Design Automation Conf. (DAC)*, 1989.

[13] Y. G. Ra, H. Kuno, and E. A. Rundensteiner, "A Flexible Object-Oriented Database Model and Implementation for Capacity-Augmenting Views", Electrical Engineering and Computer Science Dept., University of Michigan, Ann Arbor, Tech. Rep. CSE-TR-215-94, May 1994.

[14] E. A. Rundensteiner, "MultiView: A Methodology for Multiple Views in OODBs," in *Proc. of International Conference on Very Large Data Bases (VLDB)*, 1992, pp. 187-198.

[15] E. A. Rundensteiner, "Design Tool Integration Using Object-Oriented Database Views," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1993, pp. 104-107.

[16] M. H. Scholl, C. Laasch, and M. Tresch, "Updatable Views in Object-Oriented Databases," in *Proc. DOOD Conference*, Germany, Dec. 1991.