Scalable Information Management via Mediation

Michael N. Huhns and **Munindar P. Singh**

Carnot Project
Information Systems Division
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759-5398

(512) 338-3890 (FAX)
{huhns,msingh}@mcc.com

Abstract

There is an increasingly large amount of information being produced, but it is for the most part inaccessible, inconsistent, and incomprehensible. This creates a problem of information management. Previous attempts to address this problem led to local solutions that were not maintainable or scalable. We describe an approach to support the interoperation of databases, knowledge bases, applications, and interfaces. Our approach is based on the notion of mediators, but embodies techniques to exploit modularity, abstraction, and adaptation. This approach enables *open, large-scale* information management, and emphasizes *updates* as well as retrievals.

1. Introduction

Much research activity has recently been geared toward achieving interoperation within large information spaces. There are three primary prerequisites to successful interoperation. First, users, applications, and information resources need to be able to interact with the coming "information highway" autonomously, at all levels from syntactic to semantic. Second, locally autonomous systems need a nonlocal view, because joins, integrity constraints, data replication, resource preferences, and knowledge about where help or information can be found imply integration into a semantic space that spans wide-area networks. Third, they must be able to interoperate while avoiding the complexity problem of n^2 possible interface relationships among n systems. Our previous work deals with the third prerequisite; here we focus on the first two.

Specifically, we address three major problems that extant approaches do not fully handle:

1. ***Updates.*** Most related research addresses information retrieval only (e.g., [Arens et al. 93]). Updates are qualitatively more complex than retrieval. You can never violate a semantic constraint by just reading data (though an attempt to do so may violate some security or other policy constraint). Updates can, however, cause integrity constraints to be violated. Consequently, they call for a much more elaborate framework. Even in traditional homogeneous databases, updating views can be tricky; in heterogeneous systems, this problem is greatly exacerbated.
2. ***Scalability.*** How does one add a new component (e.g., database, user interface, or application) to an already large information space and have it interoperate effectively? This is the scalability problem, and there are multiple facets to its solution:
    a. ***Modularity.*** We attain modularity via a combination of distribution and mediation. In a broad sense, this is similar to other approaches; however, our specific architecture is unique.
    b. ***Abstraction.*** We define a hierarchical structuring of mediators, where mediators at higher levels in the structure maintain abstract views of lower-level components of the information space.
    c. ***Incremental expansion.*** We enable the coexistence of new applications with old ones, thereby facilitating a graceful introduction of technology while protecting past investments.
3. ***Heterogeneity of data and transaction models.*** True interoperation requires resources designed with different data and transaction models to be coherently integrated. Instead of proposing yet another such model [Elmagarmid 92], we have built the functionality by which several of them can be uniformly integrated. Different transaction models are usually closed in that they cannot be extended with the features of other models. Our approach does not commit to any single such model, but can express any of them succinctly and combine them in any way desired. Some related aspects of our approach are discussed in [Attie et al. 93] and [Huhns et al. 93].

To ground our discussions of scalability and updates, imagine the following scenario: a large brokerage house has 1000 brokers selling securities. The brokers each have a workstation or PC where they have created personal databases to record information about their clients. They store

additional data in spreadsheets that they create locally. It is reasonable to assume that the computers are interconnected, but it is likely that the semantics and syntax of each database and each spreadsheet are different. The database systems might, for example, be Paradox, Oracle, and DB2. The tables might refer to "customers," instead of "clients." The data might be stored "last-name-first" or "last-name-last." In such a diverse environment, how can

- a manager obtain information about the overall progress of the salesforce?
- the manager download new incentive tables for the brokers, when they each might represent the tables differently?
- a broker coordinate with another broker to avoid interference? It would be embarrassing if two brokers simultaneously recommended that a client buy and sell the same security.
- the manager remove from the databases all accounts that have been inactive for more than six months. This requires processing temporal interresource constraints.

Following [Wiederhold 92], we base our approach on mediators—for knowledge bases, applications, interfaces, and databases. We present a way of structuring mediators individually and collectively so that the three desiderata given above can be met. Of all the different functions that mediators can perform, we focus on workflow management as an interesting and important one well-suited to our approach. Section 2 discusses this problem and our overall approach. Section 3 presents our mediator architecture in some detail. Section 4 gives an example session of applying our approach to integrate resources within and among enterprises. Lastly, section 5 discusses applications in transaction processing in heterogeneous environments.

2. Technical Approach

2.1. The Problem: Workflow Management

The problem of information management can be cast in terms of the general problem of managing work—in government and corporate offices, design labs, factories, and so on. No one accesses information without a purpose and the purpose is best understood in the context of the overall enterprise activities. Therefore, instead of discussing information resource integration in the abstract, we approach it with a view to enabling business process integration.

Business processes, or workflows, are the structured activities that take place in the information systems of typical enterprises. Briefly, workflows consist of *tasks*, appropriately structured, where a task is a useful computation. The tasks that are of particular interest are database transactions, but other computations, e.g., those that generate visualizations, can be presented in the same framework. These activities frequently involve several database systems, user interfaces, and application programs. Traditional database systems do not support workflows to any reasonable extent. Usually, human beings must intervene to ensure the proper execution of workflows.

The activities that comprise a workflow of interest are typically already being carried out in the given organization. However, they are usually carried out by hand, with people intervening at crucial stages to ensure that the necessary tasks are done and that organization-wide consistency constraints are enforced. The semantics is supplied by the people or is implicitly encoded in

different business procedures. For instance, when an order is received, it is first entered, and then numerous decisions are made about it. These decisions typically involve accessing several information resources within an enterprise and possibly some outside of it. For example, a request to buy stock against a money account requires that the authorization be verified, the account numbers be validated, and the source account be tested to have the required balance. External sources of information would be accessed for other requests, such as loan applications, where a credit bureau's databases might be consulted to determine the credit worthiness of an applicant.

Integrating existing systems is much harder than designing them afresh. Many systems, especially those based on mainframe architectures, allow data to be accessed only through arcane interfaces. The systems and their interfaces cannot be easily modified, and our work assumes they cannot. This is because of two main reasons:

- the complexity of the programming effort that would be required to achieve any modifications, and
- the constraint that older applications continue to run as before, since they typically have a wide user base that relies heavily upon them.

Thus, the integration must permit newly developed applications to coexist with previous applications. This is a major constraint.

2.2. General Approach

Most importantly, our approach supports a new methodology for information systems design. Currently, information systems are constructed monolithically with a tight coupling among the interface, the application program, and the database (see Figure 1). In many cases, the three components are inseparable. This coupling makes the systems difficult to maintain, cumbersome to enhance, and expensive to reengineer as new technology becomes available. Collectively, these problems contribute to the formation of what are pejoratively called "legacy systems."

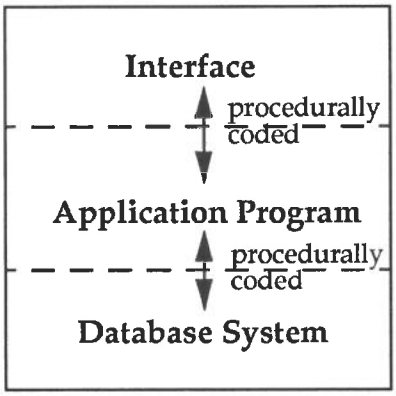

Figure 1: Model (simplified for illustrative purposes) of the tight coupling among the three major components of most current information systems.

We loosen this coupling by introducing mediators between the three components (see Figure 2). Some of the mediators function as "wrappers," i.e., protocol converters. But, in general, they can

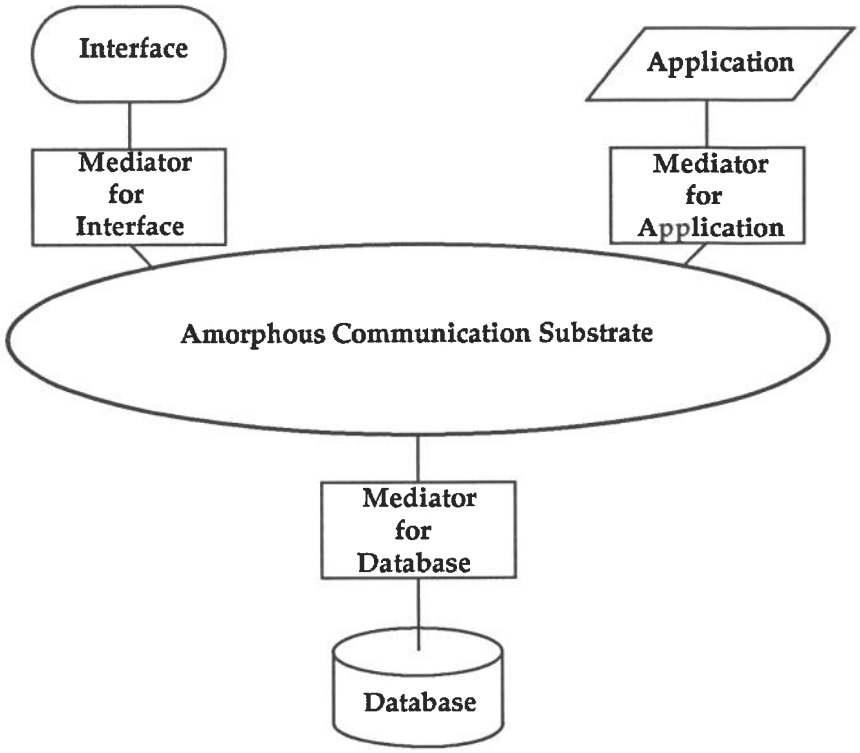

Figure 2: Information system components decoupled by the use of mediating agents.

(1) update, create, and delete information, not just query it, and (2) learn models of their own component and of other mediators. This yields modular systems that can more readily be maintained, enhanced, or reengineered.

3. Mediator Architecture

In our approach, mediators,

- translate semantics among different ontology domains,
- manage workflows, and
- enforce integrity and security constraints.

As discussed above, the properties of modularity, abstraction, adaptation, and incremental expansion are highly desirable. Our challenge, then, is to design a mediator architecture that

- is scalable,
- is extensible,
- is easy to implement and maintain,
- is reusable,
- does not violate the autonomy or semantics of integrated systems, and
- does not impose an unbearable performance penalty.

These might appear to be conflicting requirements. However, we motivate and present a design that meets the above criteria. We first discuss the structuring of each mediator and then the structuring of collections of them.

3.1. Layering of Functionality

Each mediator has a layered architecture, as shown in Figure 3.

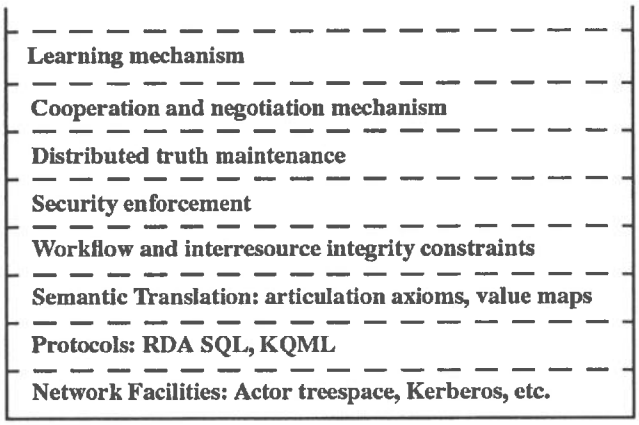

Figure 3: The Layered Architecture of a Generic Mediator

Not all layers may be present in each mediator. The lower layers handle the syntactic aspects of mediation; the upper layers the semantic and adaptation-oriented aspects. Consequently, the lower layers are easily reused and may be composed from preexisting pieces; the upper layers require human intervention, hopefully limited, in their construction.

For reasons of code reusability and testing, we implement the different layers of a mediator as different entities—actors in Carnot's Extensible Services Switch. However, we prefer to think of an entire set of such instantiated actors are one mediator because conceptually it performs as a logical unit—a mediator of a single resource.

The structuring aspects of our approach that we describe here apply to each layer of the mediators. Here we concentrate on the interresource aspects, since that is the most interesting semantically.

3.2. Locality in Mediators

We hierarchically structure collections of mediators. The lowest level of mediators are associated with individual resources and applications. Semantics relating them to the rest of the information space are captured and maintained by *local domain experts*. Higher level mediators manage interresource constraints for relating lower level systems, and common ontologies to attach a semantics to those constraints. Local domain experts can construct these ontologies incrementally, using other parts of Carnot, thereby making those ontologies extensible. This approach is scalable, because it is based on the *local* construction and maintenance of mediators.

Our approach is bottom-up and resource-driven. It respects the organization of resources present in any enterprise: for example, individual databases, although usable by themselves, are often linked to other databases within a corporate division, to resources within an entire corporation, and

increasingly nowadays to information resources in other corporations. The bottom-up approach assures not only that the implementation can be systematically constructed and tested, but also that constraints at a lower-level are never violated in favor of higher-level constraints. When higher-level constraints, such as those pertaining to intercorporate exchange of data, conflict with lower-level constraints, then the latter take precedence by default: however, such conflicts can and should be resolved through negotiation. For example, an intercorporate constraint might be that `taxable income = capital gains + (dividend per unit * number of units)`, while an intracorporate constraint might be that `income = dividend per unit * number of units.`

For each resource, we postulate a mediator that captures the constraints local to that resource. The schema of a database, its integrity constraints, and its transaction suite are in this category, as are security and access policies. In general, a mediator should be loyal to its own enterprise and not provide services to external entities that they are not authorized to access.

3.3. Scope Hierarchies

The requirements on mediators motivated above may appear to conflict. On the one hand, distribution is clearly crucial for scalability, extensibility, and to preserve local autonomy. On the other hand, the need to avoid significant performance or maintenance costs is suggestive of a hierarchical organization in which unnecessary interactions are avoided at runtime, and changes in the structure of a local system can be propagated as needed to other mediators.

Control hierarchies are antithetical to distributed computing. They create bottlenecks and hot spots for performance, and make systems susceptible to the failure of the controlling node. These limitations subvert one of the key advantages of distributed computing. Indeed, as we describe below, control hierarchies for mediators lead to additional complications that cannot be resolved without violating the autonomy of existing applications.

We resolve this dilemma through a distributed architecture in which mediators can deal with other mediators on an even footing. Additionally, we also attach a hierarchical structure to these mediators. However, the hierarchy in question does not concern control: it only concerns *scope*.

The task of creating the initial local mediators can be carried out by different resource coordinators almost independently. Typically, however, there are additional resource constraints that pertain to multiple resources. Such constraints may be known from the beginning or may emerge later through experience. Rather than attempt to capture these constraints directly in one or some of the local mediators, we define another mediator whose scope includes the relevant resources. The scopes of the different mediators thus nest in a natural manner.

The new, high-level mediator can be seen as a descriptive construct, i.e., as a virtual mediator. Service requests from without can be directed to this virtual mediator as if it mediated for a monolithic resource with complex properties and constraints. This reflects the nature of business deals in general. Enterprises as a whole provide services and enter into contractual agreements. Their internal structure, though important, is irrelevant or hidden from without.

A control hierarchy creates higher-level mediators that control predefined lower-level ones. They determine the requests to be made to different lower-level mediators to maintain high-level

constraints. Direct access to the predefined lower-level mediators must be disabled; otherwise, high-level constraints can be easily violated (see Figure 4). This debilitating limitation is not

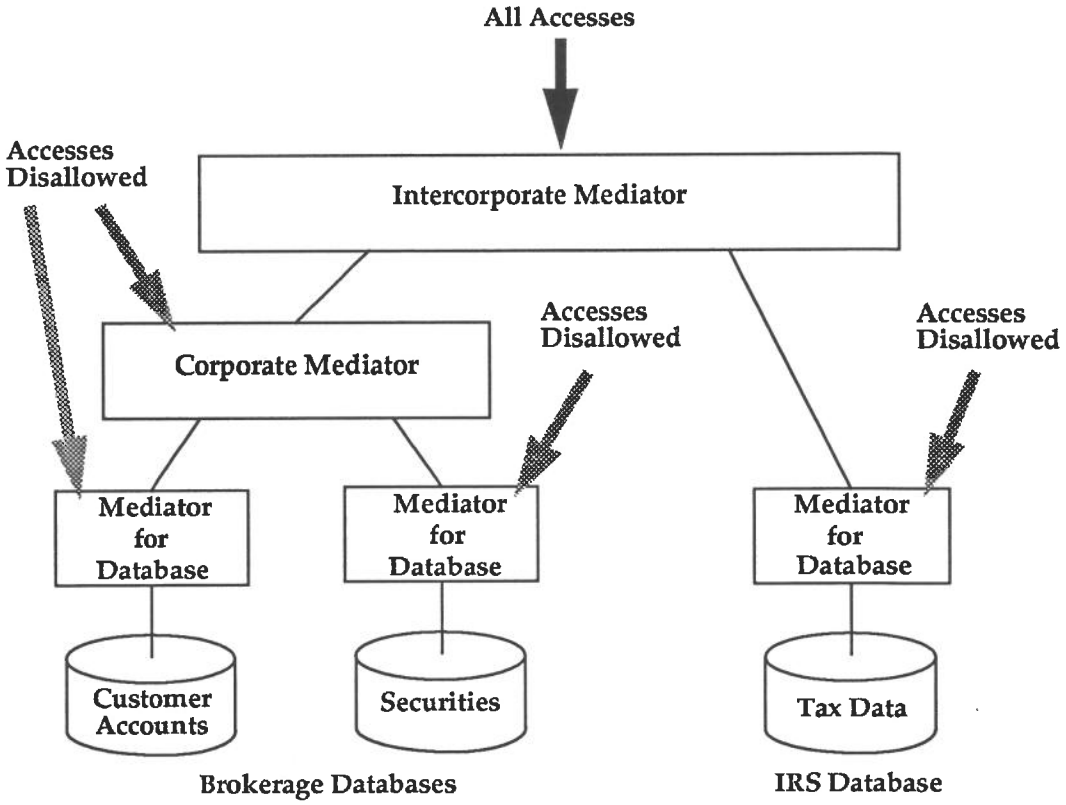

Figure 4: Control hierarchy

usually noted, since most approaches are restricted to information retrieval. As stated above, retrieval cannot cause semantic constraints to be violated. Hence, all read accesses can be safely permitted. In general, however, all requests must be funneled through the higher-level mediators.

By contrast, a scope hierarchy creates higher-level mediators that add constraints to the lower-level ones. All requests (physically) go to the lower-level mediators, which handle their old constraints as well as the new constraints. In this way the new higher-level mediator is distributed over the predefined lower-level mediators. Old applications can thus continue to issue requests directly to the lower-level mediators. Of course, if the requests issued by old applications violate new constraints, then such violations would be detected as easily as for new applications. Old applications would need to be disabled in such cases, as indeed they should be.

We should emphasize that scope hierarchy is not the same as the layering of functionality in a mediator. The latter has to do with modularizing a design so that, for example, details of database access are kept separate from a constraint reasoner. This form of layering is an instance of good software practice and simplifies implementation. The form of layering employed in our architecture was discussed in section 3.1.

Hence, our architecture achieves the goals of scalability, extensibility, and local autonomy through distribution, in a manner in which no inessential constraints are imposed. Only the necessary

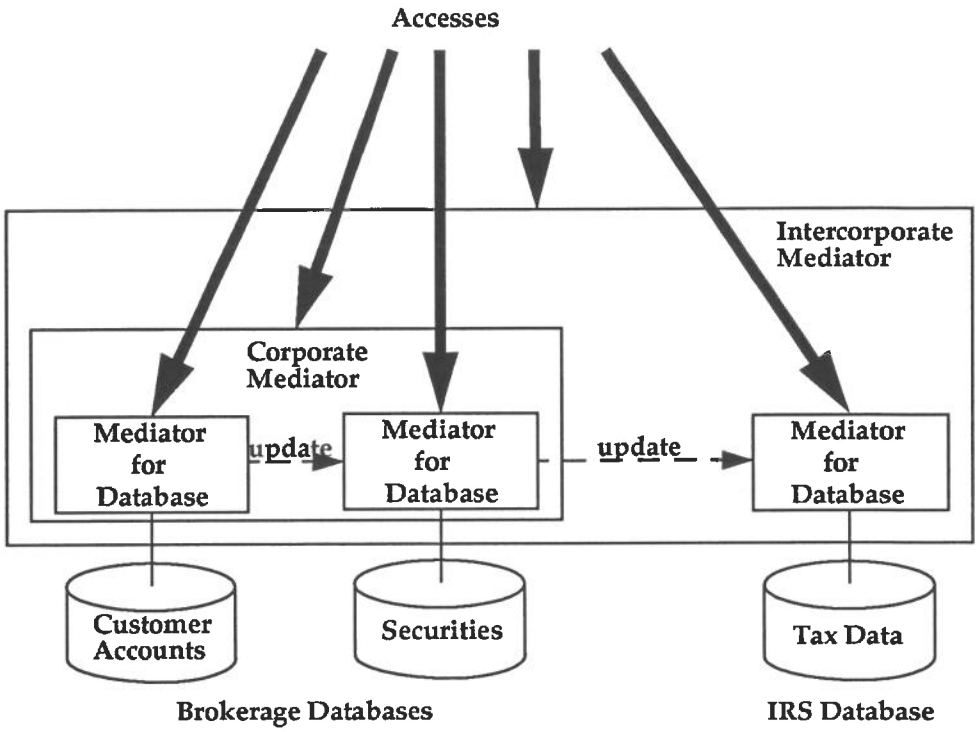

Figure 5: Scope hierarchy

interactions among mediators are required, and existing applications are affected just if they conflict with new constraints. Moreover, the mediators can be locally maintained and their different layers independently reengineered. Consequently, mediators can adapt through learning without wreaking havoc on the system as a whole.

4. An Example Session

We now describe an example session of integration using our approach.

4.1. Intracorporate Integration.

Assume that the customer account and the securities databases are already in use. The former gives the number of units of each security owned by each account; the latter gives the total number of units of each security managed by the entire brokerage firm. The accounts database is accessed by different brokers; the securities database is accessed by a VP to determine how much of the firm's business is dependent on different securities and to determine and adjust the risk for the mutual funds managed by the firm.

These databases are to be integrated to enforce the obvious semantic constraint between them. First, the administrators for the two databases must integrate their local schemas with a common ontology, which must be created if not already present. Then, the administrators are ready to capture the interresource integrity constraint. They begin by stating that constraint in terms of the common ontology and adding it to a new corporate mediator.

They then fold the constraint back into the local schemas of the two databases. The administrators also capture the temporal aspects of the constraint, namely, the frequency at which it must be validated. If it were required to be exactly maintained at all times, no feasible solution would exist in a heterogeneous environment. Fortunately, that is typically not the case. The administrators decide that whenever an update is performed on the accounts database, the mediator for the accounts database would send a corresponding transaction to the securities database. This update would eventually be executed. Thus, all existing applications continue to execute unchanged.

4.2. Intercorporate Integration.

Later, it is decided at the corporate level that certain information from the accounts and securities databases will be released to the Securities Exchange Commission. The SEC system administrators integrate their database with the common ontology, enhancing the common ontology as necessary. The additional constraint is added to a newly constructed intercorporate mediator. The constraint is folded back into the schema of the corporate mediators and then on to the local schemas of the involved databases. The necessary workflow constraints are set up as before. Consequently, all applications continue to be usable. Periodically, updates are sent from the brokerage database to the SEC database to maintain consistency of the two databases. No new bottlenecks are introduced into the brokerage's information system.

In this example, after the interresource constraint was acquired, it was folded automatically into the applicable local schemas. However, the actual cross-system updates to be performed and when to perform them also need to be determined. These are a kind of flow constraint determined by reasoning about available actions and information flow in an integrated system. Such reasoning can be extremely complex. Our aim has been to design a tool that can assist humans in this activity, but not replace them. As different kinds of reasoning appear more and more useful, we augment the reasoning component of our tool so that it can be of increasing assistance.

5. Relaxed Transaction Processing

Transaction processing provides a related, but more complex, application for our approach. Classical transaction processing in databases deals with executing access and update tasks on a single database. Such tasks are traditionally assumed to have the so-called ACID properties:

- *atomicity*, where each task happens either fully or not at all;
- *consistency*, where each task takes the database from a consistent state to a consistent state;
- *isolation*, where the intermediate results of a task are not visible to another task; and
- *durability*, where the changes caused by a task are persistent.

These assumptions help simplify transaction management considerably [Gray & Reuter 93]. However, they are overly restrictive in loosely-coupled heterogeneous environments. For example, one of the ways in which ACID tasks may be coordinated is through mutual commit protocols, which ensure that either all of a given set of tasks commit or none do. Such protocols, e.g., the two-phase commit protocol, are notoriously inefficient when executed over networks. Further, to execute such a protocol, one requires access to the internal states of a transaction, such as the

precommit states (when it is internally ready to commit, but is awaiting permission from the transaction manager to do so). Most commercial database systems do not provide access to such internal states, thereby making direct implementations of commit protocols difficult.

The ACID properties are naturally realized when the correctness of database transactions is characterized through some purely syntactic or structural criterion, such as serializability [Bernstein et al. 87]. However, serializability cannot be efficiently implemented in distributed systems whose component systems are autonomous. Following [Garcia-Molina & Salem 87], we characterize correctness criteria semantically, rather than syntactically. This simplifies coordination requirements, though at the cost of a deeper domain model. Thus, we can replace mutual commit by optimistic commit protocols. If need be, we undo the effects of incorrectly committed tasks by *compensating* them in a domain-dependent manner.

Figure 6 shows an example graph with tasks for deleting a booking from a travel agent database,

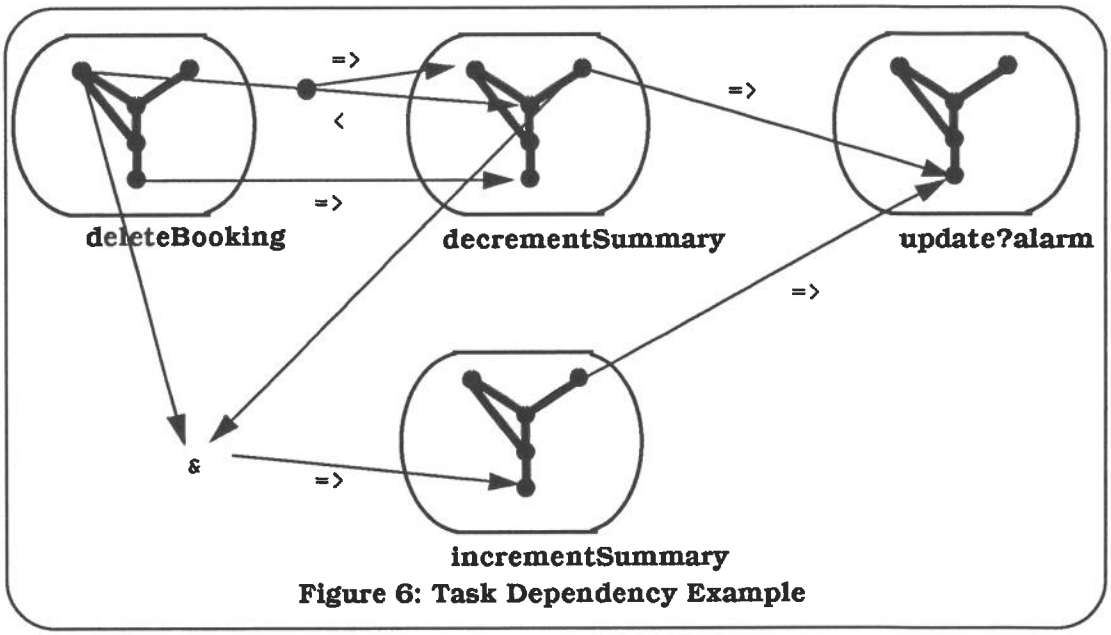

Figure 6: Task Dependency Example

incrementing and decrementing a summary of bookings, and updating an alarm if the number of bookings falls below a threshold. Each task is modeled as a finite-state automaton whose nodes are events significant to the execution of the task. For example, the existence constraint between the start events of the delete and decrement tasks indicates that the decrement task be initiated whenever the delete task is. The conjunctive constraint (&) indicates a compensating execution of the increment summary in the contingency where the delete fails and the decrement commits.

Typically, to maintain the above constraints, a script would have to be written that causes the decrement subtransaction to execute when the delete subtransaction succeeds. This is no longer necessary in our approach. The application, e.g., a user interface, that generates the delete request does not need to be aware of the summary database or the constraint between deletions and decrements. It generates the delete request as always. However, the higher-level mediator whose scope includes the delete and the decrement databases introduces a constraint that at runtime triggers the initiation of the decrement subtransaction. Other constraints related to the commit or

abort of the various subtransactions execute similarly. Thus the global semantics are enforced without affecting preexisting applications.s

6. Conclusions

The concept of mediators is applied recursively within Carnot to provide applications with access to a variety of heterogeneous data and knowledge resources. Carnot has developed mediators for several kinds and varieties of database systems, including relational database systems, such as Oracle, Ingres, and Sybase, object-oriented database systems, such as Objectivity and Itasca, and text retrieval systems, such as Topic. These mediators primarily handle protocol mediation, translate database schema semantics, and manage low-level concurrency [Woelk et al. 92]. Here we discussed how more sophisticated mediators that handle workflows can be naturally implemented.

The ESS is constructed in Rosette, a high-performance implementation of an interpreter for the Actor model [Hewitt et al. 73] enhanced with object-oriented mechanisms. This interpreter has been developed during five years of research on parallel algorithms and control of distributed applications. Actors can be understood as concurrent and asynchronous units of computation [Agha 86]. Rosette is enhanced with remote evaluation, tree spaces, and virtual synchrony. We use cooperating groups of actors to implement application-driven mediated access to resources.

Scope hierarchies can be naturally implemented using ESS's treespaces [Tomlinson et al. 93a], which are a general-purpose mechanism for relaying messages anywhere on a network. Treespaces support symbolic addressing, and allow messages to be retrieved both by their content and by the recipient's name. The treespace facility pertains to the lower layers of the Carnot architecture. It is used for message addressing by the various database and expert system mediators we have already implemented.

We presented a mediator architecture in which the individual mediators are layered by functionality and organized in a manner that yields the benefits of hierarchies without losing the benefits of distribution. Our approach yields local autonomy, easy maintainability, efficiency, and *scalability*. The architecture is embedded in an interpretive, dynamic development and execution environment and provides capabilities ranging from protocol conversion to intelligent cooperation and learning. It enables the simple incorporation of new components into large information spaces. Our approach is also unique in that we develop mediation techniques for *transaction processing* on information resources, not just querying.

References

[Agha 86] Gul Agha, *Actors,* The MIT Press, Cambridge, MA, 1986.

[Ahlsen & Johannesson 90] Matts Ahlsen and Paul Johannesson. "Contracts in Database Federations," in S. M. Deen, ed., *Cooperating Knowledge Based Systems 1990,* Springer-Verlag, London, 1991, pp. 293–310.

[Ansari et al. 92] Mansoor Ansari, Marek Rusinkiewicz, Linda Ness, and Amit Sheth, "Executing Multidatabase Transactions," *Proceedings 25th Hawaii International Conference on Sys-*

tems Sciences, January 1992.

[Arens et al. 93] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock, "Retrieving and Integrating Data from Multiple Information Sources," *International Journal on Intelligent and Cooperative Information Systems*, Vol. 2, No. 2, 1993, pp. 127–158.

[Attie et al. 93] Paul C. Attie, Munindar P. Singh, Amit P. Sheth, and Marek Rusinkiewicz, ``Specifying and Enforcing Intertask Dependencies,'' *Proceedings of the 19th VLDB Conference*, 1993.

[Bernstein et al. 87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Co., 1987.

[Bukhres et al. 93] Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, and Robert Pezzoli, ``InterBase: An Execution Environment for Heterogeneous Software Systems,'' *IEEE Computer*, Vol. 26, No. 8, Aug. 1993, pp. 57–69.

[Buneman et al. 90] O.P. Buneman, S. B. Davidson, and A. Watters, "Querying Independent Databases," *Information Sciences*, Vol. 52, December 1990, pp. 1–34.

[Cannata 91] Philip E. Cannata, "The Irresistible Move towards Interoperable Database Systems," *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 7–9, 1991.

[Elmagarmid 92] Ahmed K. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, Morgan Kaufman, San Mateo, CA, 1992.

[Finin et al. 92] Tim Finin, Don McKay, and Rich Fritzson, "An Overview of KQML: A Knowledge Query and Manipulation Language," University of Maryland Computer Science Technical Report, March 1992.

[Garcia-Molina & Salem 87] Hector Garcia-Molina and K. Salem, "Sagas,'' *Proceedings of ACM SIGMOD Conference on Management of Data*, 1987.

[Gray & Reuter 93] Jim Gray and Andreas Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufman, San Mateo, CA, 1993.

[Heimbigner & McLeod 85] Dennis Heimbigner and Dennis McLeod, "A Federated Architecture for Information Management," *ACM Transactions on Office Information Systems*, Vol. 3, No. 3, July 1985, pp. 253–278.

[Huhns et al. 93] Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Wei-Min Shen, Munindar Singh, and Philip Cannata, ''Integrating Enterprise Information Models in Carnot,'' *International Conference on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, The Netherlands, June 1993.

[Huhns & Singh 94] Michael N. Huhns and Munindar P. Singh, ``Automating Workflows for Service Provisioning: Integrating AI and Database Technologies,'' *10th IEEE Conference on Artificial Intelligence Applications*, San Antonio, Texas, March 1994.

[Klein 91] Johannes Klein, "Advanced Rule Driven Transaction Management," *Proceedings of the IEEE COMPCON*, 1991.

[Tomlinson et al. 93b] Christine Tomlinson, Paul Attie, Philip Cannata, Greg Meredith, Amit Sheth, Munindar Singh, and Darrell Woelk, "Workflow Support in Carnot,'' *IEEE Data Engineering*, 1993.

[Wiederhold 92] Gio Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, March 1992, pp. 38–49.

[Woelk et al. 93] Darrell Woelk, Paul Attie, Philip Cannata, Greg Meredith, Munindar Singh, and Christine Tomlinson, ''Task Scheduling Using Intertask Dependencies in Carnot,'' *ACM SIGMOD*, 1993.