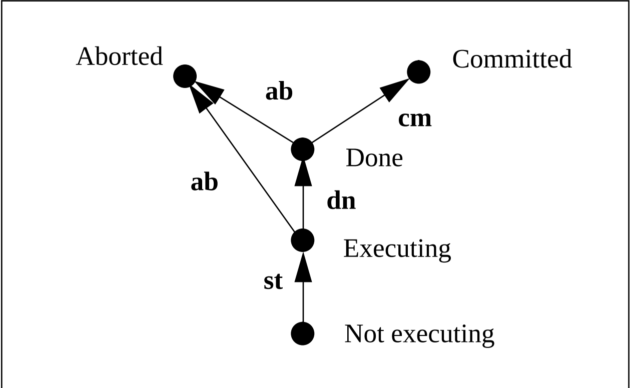


Figure 3: An Example Task State Transition Diagram

1. e_1 -> e_2: If e_1 occurs, then e_2 must also occur. There is no implied ordering.
2. $e_1 < e_2$: If e_1 and e_2 both occur, then $e1$ must precede $e2$.

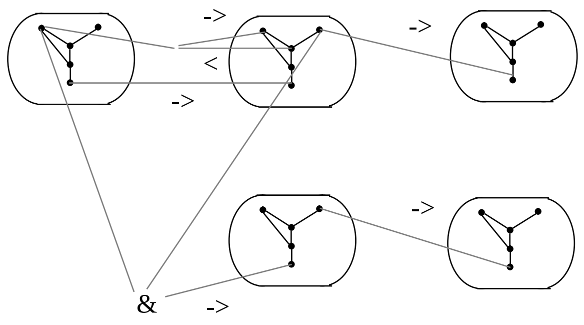
Examples of execution dependencies defined in the literature include:

1. Commit Dependency [2]: Transaction *A* is commit-dependent on transaction *B*, iff if both transactions commit, then *A* commits before *B* commits. Let the relevant significant events be denoted as cm_A and cm_B. This can be expressed as $cm_A < cm_B$.
2. Abort Dependency [2]: Transaction *A* is abort-dependent on transaction *B*, iff if *B* aborts, then *A* must also abort. Let the significant events here be ab_A and ab_B, so this can be written ab_B -> ab_A.

The relationships between significant events of a task can be represented by a task state transition diagram which is an abstract representation of the actual task that hides irrelevant details of its sequential computations. Execution of the event causes a transition of the task to another state. Figure 3 shows an example task state transition diagram taken from [6]. From its initial state (at the bottom of the diagram), the task first executes a start event (**st**). Once the task has started, it will eventually either abort, as represented by the **ab** transition, or finish, as represented by the **dn** transition (for “done”). When a task is done, it can either commit, i.e., make the **cm** transition, or abort, i.e., make the **ab** transition.

Using the state transition diagrams and significant events defined above, we can represent the travel agent application described in the previous section as shown in Figure 4 . The intertask dependencies are shown as “links” between significant events of various tasks. For example, the dependency (ab(dB) < dn(dS)) -> ab(dS) states that if the dB task aborts before the dS task is done executing, then the dS task will also abort.

The formal specification of these dependencies using a



st(dB) -> st(dS)
cm(dS) -> st(u?a)
cm(iS) -> st(u?a)
(ab(dB) & cm(dS)) -> st(iS)
(ab(dB) < dn(dS)) -> ab(dS)

Figure 4: Dependencies Between Significant Events

Computation Tree Logic [5] and the implementation of a scheduler to enforce these dependencies is described in [1]. The scheduler is implemented in the concurrent actor language Rosette. We are continuing to enhance this approach by exploiting results in action logics [7] and to experiment with the construction of advanced transaction models.

Acknowledgments

We would like to thank Marek Rusinkiewicz of the University of Houston for his contributions to this work.

References

[1] Attie, P., M. Singh, A. Sheth, and M. Rusinkiewicz. “Specifying and Enforcing Intertask Dependencies”. submitted for publication, January, 1993..

[2] Chrysanthis, P. and K. Ramamritham. “ACTA: The SAGA Continues”. Chapter 10 in [4].

[3] Elmagarmid, A., Y. Leu, W. Litwin, and M. Rusinkiewicz. “A Multidatabase Transaction Model for Interbase”. *Proceedings of the VLDB Conference, August*, 1990.

[4] Elmagarmid, A., editor. **Database Transaction Models for Advanced Applications**, Morgan Kaufmann, 1992.

[5] Emerson, A. and E. Clarke. “Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons”. *Science of Computer Programming*, vol.2, 1982, 241-266.

[6] Klein, J. “Advanced Rule Driven Transaction Management.” *Proceedings of the IEEE COMPCON*, 1991.

[7] Pratt, V.R. ”Action Logic and Pure Induction”. Logics in AI: European Workshop JELIA ‘90, LNCS 478, Editor: J. van Eijck, Springer-Verlag”, pp. 97-120, 1990.

[8] Woelk, D., P. Cannata, M. Huhns, W. Shen, and C. Tomlinson. “Using Carnot for Enterprise Information Integration”. *Second International Conference on Parallel and Distributed Information Systems*. January, 1993. pp. 133-136.

.

(finite state machine) for each task, and a set of constraints among these automata that control the execution of each automaton based on the state of the other automata.

3. Specifying and Enforcing Intertask Dependencies

A demonstration application has been developed at MCC that focuses on interoperability among database servers. This application originally was developed for the SQL Access Group Interoperability Demo in July 1991. SQL Access Group is a consortium of leading software and hardware companies working together to develop a standard SQL interface for database management systems and standard protocols for interoperability among clients and servers using the ISO Remote Database Access (RDA) protocol.

The MCC database server for the demo was an Itasca object-oriented database system with an Object SQL interface, a subset of which provides standard SQL functionality. The Object SQL interface was written by the Carnot project and is an enhancement to the existing Itasca query language. Itasca is a commercial product marketed by Itasca Systems that is based on the MCC ORION prototype. The MCC client was an application developed using the Carnot Graphical Interaction Environment (GIE). The databases used for the SQL Access Group Demo contained information on bookings for air travel, hotels, and entertainment for a travel agency. Each of the database servers handled data for a specific location. Each of the clients accessed one or more of the servers to schedule trips, generate reports, or just browse the databases. The MCC client application consisted of a graphical interface developed using the Carnot Graphical Interaction Environment (GIE), which allowed the user to click on an icon to generate queries. Each query was expanded to include access to all semantically equivalent and relevant information resources as described in [8].

The application has now been enhanced to include a separate Itasca database containing summary information on the number of bookings per travel agent. When an agent's booking is deleted from the booking database, the booking count for the agent should be decremented in the summary database. If the count falls below some specified value, an icon should flash red.

These semantics can be captured with an integrity constraint that the number of rows in the booking relation should equal the number of bookings stored for that agent in the summary relation. The maintenance of this constraint can be assured by executing updates to each database as atomic multidatabase transactions using a protocol such as two-phase commit. However, the database systems we are using do not necessarily provide visible two-phase commit facilities.

Instead, we may assume that the interdatabase integrity is maintained by executing separate tasks that obey the appropriate intertask dependencies. Realistic dependencies for this example state that if a delete task on the booking relation commits, then a decrement-summary task should also commit. Furthermore, if a delete task aborts, while its associated decrement-summary task commits, then we must restore consistency by compensating for the spurious decrement. We do this by executing an increment-summary task. Figure 2 shows the tasks involved in this example; dB, dS, iS, and u?a denote the delete-booking, decrement-summary, increment-summary, and update-alarm tasks, respectively.

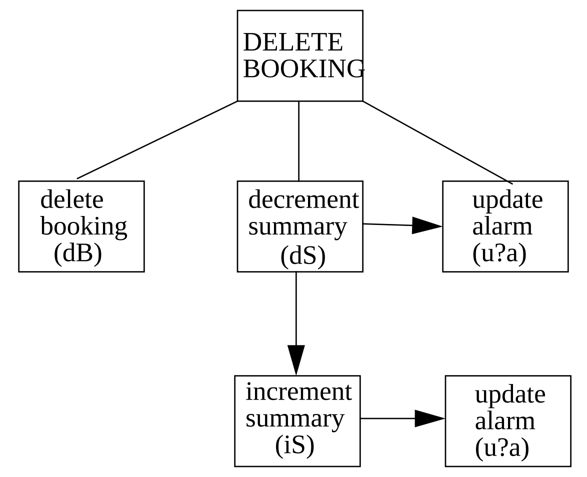
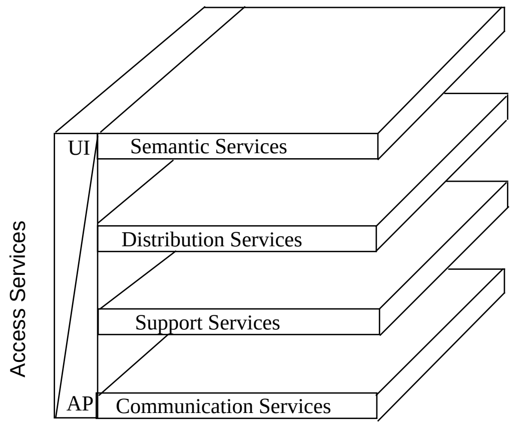


Figure 2: Task Graph for the Delete Booking Example

There are dependencies among the task boxes in Figure 2. These dependencies determine the allowable orderings of execution of the tasks. There are three potential strategies for controlling the ordering of execution of these tasks. The *first strategy* is to give a name to this pattern of tasks and dependencies and implement a transaction scheduler specifically for this pattern. This is the strategy normally used for traditional two-phase commit implementations. The *second strategy* is to give a name to each of the dependencies and implement a scheduler which interacts with tasks based on a programmed semantics for the named dependency. An example is a scheduler for traditional flat transactions (where all subtransactions must either all commit or all abort) where each subtask has a Commit Dependency (see definition below) on every other subtask. The third strategy is to identify the significant internal states of each subtask, identify a few simple types of dependencies among these internal states, and implement a scheduler which enforces these dependencies. We have chosen the third strategy. This choice is described further below.

The specification and enforcement of intertask dependencies has recently received considerable attention [2,3,4,6]. Following [2] and [6], we specify intertask dependencies as constraints on the occurrence and temporal order of certain significant events specified on a per-task basis. Klein has proposed the following two primitives [6]:

Access Services -
•2D & 3D Graphical Interaction Environment
•Deductive Computing
•Application Frameworks



Semantic Services -
•Enterprise Modeling and Model Integration
•Knowledge Discovery
•Application Dredging

Distribution Services -
•Relaxed Transaction Processing
•Communicating Agents
•Workflow Manager
•Legacy System Access
•Distributed Administration
•Declarative Resource Constraint Base

Support Services -
•Extensible Services Switch •RDA •TP •IRDS •ORB
•X.500 •X.400 •Security •SNMP •CMIP •EDI

Communication Services -
•OSI •Internet •X.25 •SNA •DCE •Atlas
•SMDS •Frame Relay •FDDI •BISDN

Figure 1. Carnot Architecture

model declaratively within the global context and to construct bidirectional mappings between the model and the global context.

The knowledge discovery methods provide tools to discover useful patterns and regularities from information resources and check consistency between information and corresponding models. Application dredging provides tools and methods for extracting information (the initial focus being database dependency information) from application code and artifacts, guided by expectations about the nature of the information and its embedding in the application.

The ***access services*** provide mechanisms for manipulating the other four Carnot services. The access services allow developers to use a mix of user interface software and application software to build enterprise-wide systems. Some situations (such as background processing) utilize only application code and have no user interface component. In other situations, there is a mix of user interface and application code. Finally, there are situations in which user interface code provides direct access to functionalities of one or more of the four services.

The user interface software supported by Carnot includes a 2D and 3D model-based visualization facility and an object-oriented deductive computing environment, LDL++, that is fully integrated with C++ and optimized for recursive query processing.

2. Relaxed Transaction Processing

A number of sponsors of the Carnot research have started development of applications using the Carnot prototype, including Eastman Kodak, Boeing Computer Services, Bellcore, and Ameritech. The focus of these applications varies from an emphasis on heterogeneous database access to an emphasis on more generalized workflow processing. The recurring theme in each of these applications, however, is the requirement to simplify the development of distributed, enterprise-wide applications through the use of tools that first capture relationships and constraints among information resources and then automatically expand queries using these relationships and automatically enforce constraints when updates occur.

This theme is particularly strong in the area of transaction processing in environments where heterogeneous, distributed, and autonomous systems are required to coordinate the update of the local information under their control. In such environments, it may not be possible, or desirable, to maintain the ACID properties of a traditional transaction, which require using a protocol such as the two-phase commit protocol [4]. Furthermore, the inter-resource dependencies, consistency requirements, and contingency strategies for the enterprise may require a complex set of real time decisions to be made concerning when and where updates are made, and which take into account the effect of failures at local systems. Instead of encoding these decisions into individual applications, we represent them declaratively as a separate set of rules that can have an impact on many applications

These rules not only express consistency requirements, but can also capture advanced transaction models, such as Sagas, which are needed to control the actual execution of distributed transactions. The advanced transaction models are captured by specifying a set of tasks, an automaton

Task Scheduling Using Intertask Dependencies in Carnot

Darrell Woelk*, Paul Attie, Phil Cannata**, Greg Meredith,
Amit Sheth***, Munindar Singh, Christine Tomlinson
MCC
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

The Carnot Project at MCC is addressing the problem of logically unifying physically-distributed, enterprise-wide, heterogeneous information. Carnot will provide a user with the means to navigate information efficiently and transparently, to update that information consistently, and to write applications easily for large, heterogeneous, distributed information systems. A prototype has been implemented which provides services for (a) enterprise modeling and model integration to create an enterprise-wide view, (b) semantic expansion of queries on the view to queries on individual resources, and (c) inter-resource consistency management. This paper describes the Carnot approach to transaction processing in environments where heterogeneous, distributed, and autonomous systems are required to coordinate the update of the local information under their control. In this approach, subtransactions are represented as a set of tasks and a set of intertask dependencies that capture the semantics of a particular relaxed transaction model. A scheduler has been implemented which schedules the execution of these tasks in the Carnot environment so that all intertask dependencies are satisfied.

1. Overview of Carnot

Carnot has developed and assembled a large set of generic facilities that are focused on the problem of managing integrated enterprise information. These facilities are organized as five sets of services as shown in Figure 1: communication services, support services, distribution services, semantic services, and access services.

* On assignment at MCC from NCR
** On assignment at MCC from Bellcore
***Bellcore, Piscataway, NJ 08854-4182

The ***communication services*** provide the user with a uniform method of interconnecting heterogeneous equipment and resources. These services implement and integrate various communication platforms that may occur within an enterprise. Such platforms are considered to provide functionality up to the application layer of the ISO OSI reference model.

The ***support services*** implement basic network-wide utilities that are available to applications and other higher level services. These services currently include the ISO OSI Association Control (ACSE), ISO OSI Remote Operations (ROSE), CCITT Directory Service (X.500), and ISO Remote Data Access (RDA). Itasca, Ingres, and Oracle DBMS's on Unix and IBM MVS and VM DBMS's can be accessed.

An important component of the support services layer that is unique to Carnot is a distributed shell environment called the Extensible Services Switch (ESS). The ESS provides access to communication resources, local information resources, and applications at a site.

The ***distribution services*** support relaxed transaction processors (processors that appropriately manage information inconsistency) and a distributed agent facility that interacts with client applications, directory services, repository managers, and Carnot's declarative resource constraint base to build ESS workflow scripts designed to carry out some business function. The workflow scripts execute tasks that properly reflect current business realities and accumulated corporate folklore. The declarative resource constraint base is a collection of predicates that expresses business rules, inter-resource dependencies, consistency requirements, and contingency strategies throughout the enterprise.

The ***semantic services*** provide a global or enterprise-wide view of all the resources integrated within a Carnot-supported system. The Enterprise Modeling and Model Integration facility uses a large common-sense knowledge base as a global context and federation mechanism for coherent integration of concepts expressed within a set of enterprise models. A suite of tools uses an extensive set of semantic properties to represent an enterprise information