Contents

Chapter 2

Carnot Prototype

Darrell Woelk, Philip Cannata, Michael Huhns, Nigel Jacobs, Tomasz Ksiezyk, R. Greg Lavender, Greg Meredith, KayLiang Ong, WeiMin Shen, Munindar Singh, and Christine Tomlinson

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759, USA
{woelk,huhns,jacobs,ksiezyk,ong,wshen,msingh,tomlic}@mcc.com

2.1 Introduction

The Carnot Project at MCC was initiated in 1990 with the goal of addressing the problem of logically unifying physically-distributed, enterprise-wide, heterogeneous information. Carnot provides 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 that provides services for (1) enterprise modeling and model integration to create an enterprise-wide view, (2) semantic expansion of queries on the view to queries on individual resources, and (3) interresource consistency management. Carnot also includes technology for 3D visualization of large information spaces, knowledge discovery in databases, and software application design recovery.

A key technical problem addressed by Carnot is the need to simplify the development of enterprise-wide applications that access information and keep information consistent. This requires that the Carnot system maintain a semantically rich understanding of the information used to run the enterprise. This understanding, in the form of a model of the enterprise, is kept in a knowledge base that is part of the Carnot system. Of course, the real data about the operation of the enterprise are maintained in various physical resources, such as databases, files systems, and application programs.

Once a model of the enterprise has been created, the database schemas can be individually related to this model. When this step is completed, each operation on an individual database schema has an equivalent operation on the enterprise model and an operation on the enterprise model can map into operations on multiple databases.

Furthermore, business rules that in the past were embodied in application programs can be represented in the enterprise model where the Carnot system can enforce them, thus simplifying the development of new application programs. Also, as new computer hardware, database management systems, or databases are added to the enterprise, they are also individually related to the enterprise model. The result is a powerful system that enables an unlinking of applications from physical resources. Applications do not need to change as a business expands, or when two businesses merge.

The implementation of the Carnot system has required unique advances in two technology areas. First, innovative techniques for knowledge representation have been developed to capture and maintain an enterprise model and to map operations between an enterprise model and the physical databases. Creative new tools have also been developed to discover new knowledge in existing databases, code, and other artifacts. Second, a flexible, dynamic, distributed processing environment has been developed that supports the automatic generation of program scripts that execute on heterogeneous, distributed systems. The scripts control the flow of processing and can be reconfigured dynamically to respond to changes in the hardware environment or to the incorporation of additional information resources. The scripts are embedded in autonomous computing agents that can be dispatched to remote sites.

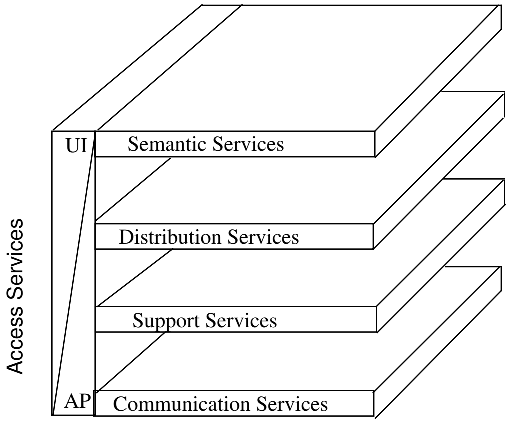
The remainder of this chapter describes the Carnot architecture, along with examples of software modules that have been developed over the last four years. Then, the use of the Carnot prototype software for heterogeneous database integration is described in more detail.

2.2 Carnot Architecture

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 2.1: communication services, support services, distribution services, semantic services, and access services.

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. Examples of such platforms include ISO OSI session and presentation layer protocols running on top of ISO TP4 with CLNP, TCP/IP via a convergence protocol, or X.25. Other possible platforms are OSF's DCE and UI's Atlas. While the communication services provide the interconnection, it is the

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
- Distributed Query 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 2.1. Carnot Architecture

services above them that allow for their effective use.

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), CCITT Message Handling System (X.400), ISO Remote Data Access (RDA). Additional interfaces include the MIT Project Athena Authentication Service (Kerberos), the ISO Transaction Processing, OMG's Object Request Broker (ORB) and Basic Object Adapter (BOA), interfaces to Information Resource Dictionary Systems (IRDS), Electronic Data Interchange (EDI), and Network Management via SNMP and CMIS.

An important component of the support services layer that is unique to Carnot is a distributed actor environment called the Extensible Services Switch (ESS) [TOML92]. The ESS provides access to communication resources, local information resources, and applications at a site. It can be used to coordinate the execution of work across a variety of heterogeneous resources, including databases, files, and other applications used in running a business. The ESS is constructed on top of a high performance implementation of an interpreter for an enhanced version of the MIT Actor language.

The distribution services add significant value to the platform-level services found in the support and communication services. In this layer, relaxed transaction processors (processors that appropriately manage information inconsistency) and a distributed agent facility interact 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 [WOEL92]. 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, interresource dependencies, consistency requirements, and contingency strategies throughout the enterprise. The declarative resource constraint base can adapt to a changing environment without modifying the application programs that manage the transactions; new applications can easily be added to the environment.

The semantic services provide a global or enterprise-wide view of all the resources integrated within a Carnot-supported system. This view, or portions of the view, can be compiled for use within the distribution services layer.

The Enterprise Modeling and Model Integration facility uses a large commonsense knowledge base as a global context and federation mechanism for coherent integration of concepts expressed within a set of enterprise models [HUHN92]. A suite of tools uses an extensive set of semantic properties to represent an enterprise information 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. Carnot's approach combines a patent-pending inductive learning method with a spectrum of ways of using knowledge, ranging from pure deductive proof to analogical reasoning.

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 [BIGE93].

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.

2.3 Using Carnot for Integration of Heterogeneous Databases

This section describes in more detail the application of the Carnot prototype software to the problem of integrating heterogeneous databases. Figure 2.2 is a representation of the Carnot prototype environment. At the top of Figure 2.2 is the Carnot runtime Information Management Infrastructure, which supports application clients, database servers, and a layer of middleware software.

In the lower left hand corner of Figure 2.2 is a representation of the Enterprise Modeling Infrastructure, which supports the integration of interfaces, applications, and databases. The results of enterprise modeling are a global dictionary containing information on physical location and types of local databases that can be accessed, descriptions of local schemas of these databases, articulation axioms that describe equivalence mappings between concepts in a local database and concepts in a global context, and a declarative resource constraint base that describes interdatabase update consistency dependencies. These results are available to the Information Management Infrastructure at runtime, as shown at the top of Figure 2.2.

In the lower right hand corner of Figure 2.2 is a representation of the Application Dredging Infrastructure. It is based on the DESIRE software prototype developed previously at MCC [BIGE93]. DESIRE is a system for software design information recovery. It includes a platform for collaborative investigations in information capture and use. A user is assisted in these investigations by custom, navigable, analyzable, hypermedia views of code and software elements. These ideas have been used within the Carnot project to recognize database queries in applications, recognize database update transactions in applications, and to extract interdatabase dependencies from applications. These dependencies can then be stored in the Carnot declarative resource constraint base, where they can be enforced for all applications.

The following sections describe the Enterprise Modeling Infrastructure and the

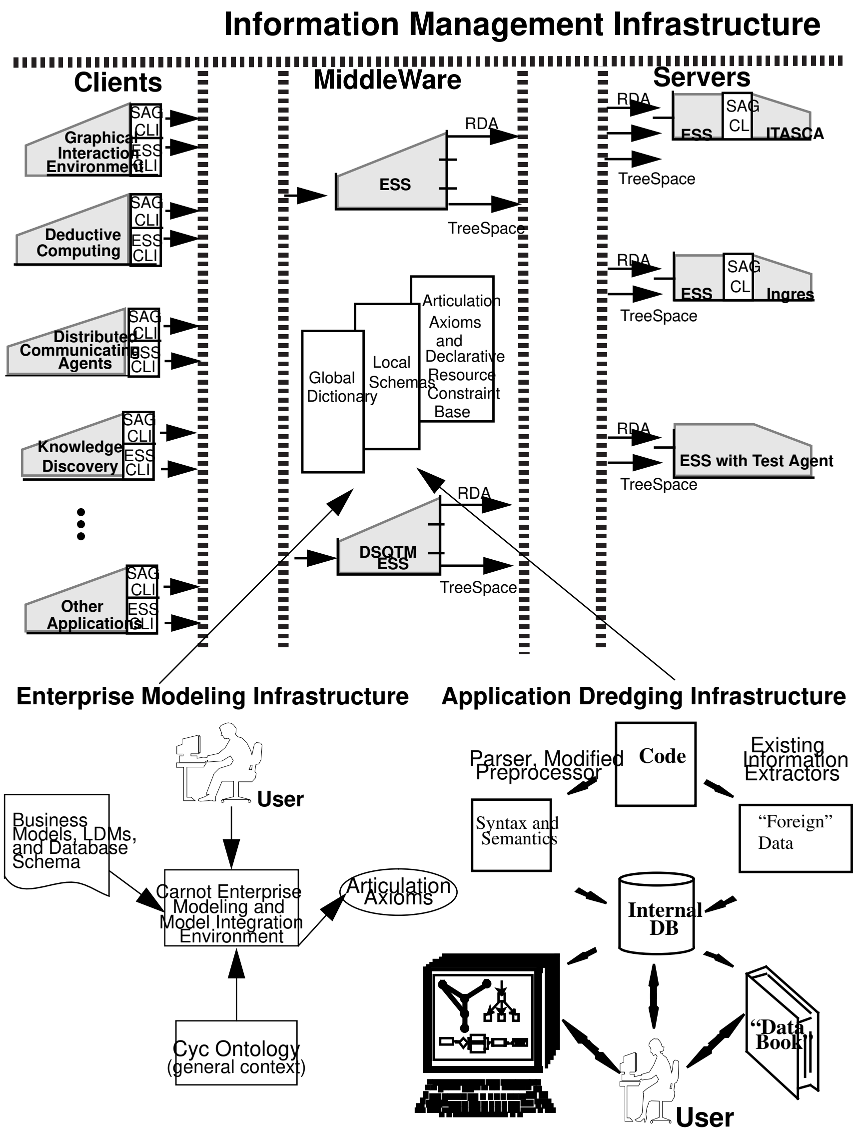


Figure 2.2. Carnot Architecture

Information Management Infrastructure in more detail. The Application Dredging Infrastructure is not described further.

2.4 Enterprise Modeling Infrastructure—MIST

Carnot has the capability to integrate separately developed information models. The models may be the schemas of databases, frame systems of knowledge bases, domain models of business environments, or process models of business operations. The method achieves integration at a semantic level by using an existing global ontology to develop semantic mappings among resources and resolve inconsistencies. This method is incorporated in a graphical integration tool, the Model Integration Software Tool (MIST). The integrated models provide a coherent picture of an enterprise and enable its resources to be accessed and modified coherently [HUHN92].

2.4.1 Approach to Model Integration

There are two general approaches to integrating models as suggested in [BUNE90]. The composite approach introduces a global schema to describe the information in the given databases. Users and applications are presented with the illusion of a single, centralized database. Explicit resolutions are specified in advance for any semantic conflicts among the databases. However, the centralized view may differ from the previous local views and existing applications may not execute correctly any more. Further, a new global schema must be constructed every time a local schema changes or is added.

The federated [HEIM85] or the multidatabase [LITW90] approach presents a user with a collection of local schemas, along with tools for information sharing. The user resolves conflicts in an application-specific manner, and integrates only the required portions of the databases. This approach yields easier maintenance, increased security, and the ability to deal with inconsistencies. However, a user must understand the contents of each database to know what to include in a query: there is no global schema to provide advice about semantics. Also, each database must maintain knowledge about the other databases with which it shares information, e.g., in the form of models of the other databases or partial global schemas [AHLS90]. For n databases, as many as n(n-1) partial global schemas might be required, while n mappings would suffice to translate between the databases and a global schema.

We base our methodology on the composite approach, but make four changes that enable us to combine the advantages of both approaches while avoiding some of their shortcomings.

1. We use an existing global context—the Cyc knowledge base [LENA90]. The schemas (models) of individual resources are compared and merged with Cyc but not with each other, making a global context much easier to construct and maintain.

2. Unlike most previous work on integration, we use not just a structural description of a local model, but all available knowledge, including (1) schema knowledge, i.e., the structure of the data, integrity constraints, and allowed operations; (2) resource knowledge, i.e., a description of supported services, such as the data model and languages, lexical definitions of object names, the data itself, comments from resource designers and integrators; and (3) organization knowledge, i.e., the corporate rules governing use of the resource.

3. We capture the mapping between each model and the global context in a set of articulation axioms: statements of equivalence between components of two theories [GUHA90]. The axioms provide a means of translation that enables the maintenance of a global view of all information resources and, at the same time, a set of local views that correspond to each individual resource. An application can retain its current view, but use the information in other resources. Of course, any application can be modified to use the global view directly to access all available information.

4. We consider knowledge-based systems (KBSs) and process models, as well as databases. A KBS may be an information resource (similar to a database), an application that accesses other resources, or a federating mechanism that serves semantically as an intermediary between applications and databases.

2.4.2 Development of Articulation Axioms using MIST

A key to Carnot's coherent integration of models is its use of the Cyc common-sense knowledge base as a global context and federating mechanism. Further, Carnot uses the knowledge representation language of Cyc to express both the static information structures and dynamic processes of an enterprise. The broad coverage of Cyc's knowledge enables it to serve as a fixed-point for representing not only the semantics of various information modeling formalisms, but also the semantics of the domains being modeled. The models can be constructed using any of several popular formalisms, such as IRDS, IBM's AD/Cycle, Bellcore's CLDM, and Andersen Consulting's Information Model. The relationship between Cyc and these formalisms is indicated in Figure 2.3.

In Carnot, the relationship between a domain concept from a local model and one or more concepts in the global context is expressed as an articulation axiom. Enterprise models are then related to each other—or translated between formalisms—via this global context by means of the articulation axioms. As a result, each enterprise model can be integrated independently, and the articulation axioms that result do not have to change when additional models are integrated.

This same technology can also be used to integrate database schemas and database application views. As the semantics are expressed formally in Cyc's logic language, they result in automatically maintainable constraints on the models. These same constraints can be extended to the databases underlying the models. In this way, high-level business rules can automatically generate DBMS-enforced

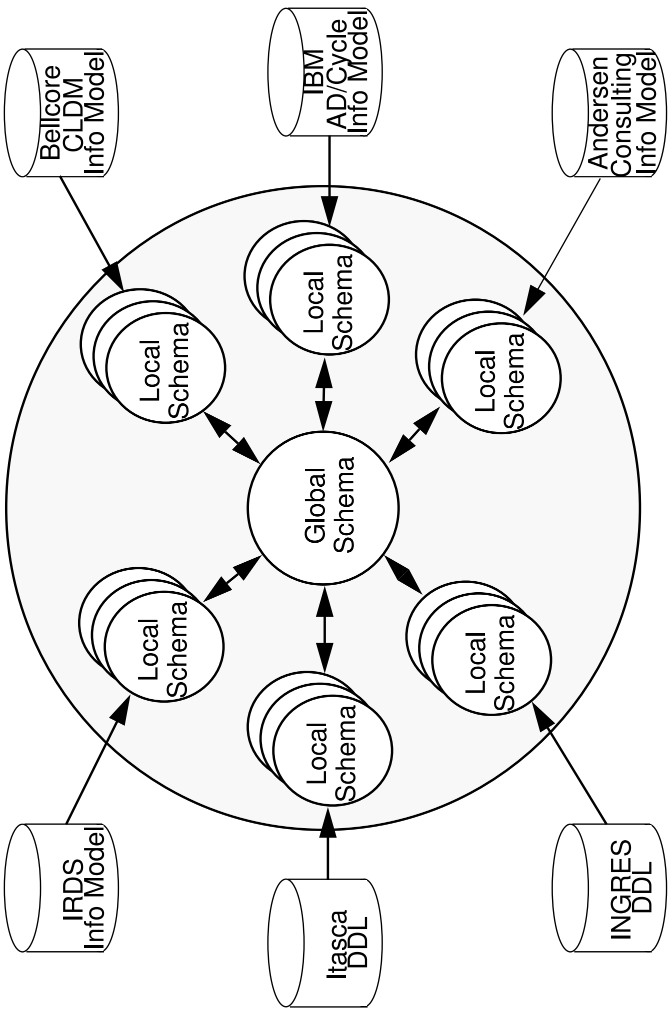


Figure 2.3. Cyc Global Context

integrity constraints. Also, besides its common-sense knowledge of the world, Cyc has knowledge about most data models and the relationships among them. This enables database transactions to interoperate semantically between, for example, relational and object-oriented databases.

The Carnot MIST tool automates the routine aspects of model integration, while clearly displaying the information needed for effective user interaction. The tool produces articulation axioms in the following three phases: representation of the model, matching of concepts, and construction of articulation axioms.

As shown in Figure 2.4, MIST displays enterprise models both before (lower right) and after (lower left) they are represented in a local context of Cyc. MIST enables the Cyc knowledge base to be browsed graphically (middle right) and textually (upper left), in order to allow the correct concept matches to be located. It enables a user to create frames in the global context or augment the local context for a model with additional properties when needed to ensure a successful match. MIST also displays the articulation axioms (upper right) that it constructs. The rest of this section describes the three phases of articulation axiom development in more detail.

Model Representation

In this phase, we represent the model as a set of frames (classes and slots) in a context created specially for it. These frames are instances of frames describing the data model of the schema, e.g., (for a relational schema) Relation and DatabaseAttribute.

We define three types of frames for representing models: (1) DatabaseSchema frames, describing schemas for different data models, (2) DatabaseObject frames, describing the major components of schemas, such as relations and entities, and (3) DatabaseAttribute frames, describing different kinds of links used to refine and relate the major components. Every schema and every one of its components (relation, attribute, etc.) is an instanceOf these types and belongs to a context characterizing that schema. The slot dBSchemaMt, defined for DatabaseSchema, is used to express the relationship between an instance of a schema and its context. Information about the usage of a resource and the functionalities it provides (e.g., management and access languages, transactions, etc.) are represented similarly, i.e., using frames such as RelationalService, ERService, RelationalDDLType, ObjectOrientedMethod, and ERTransactionType.

Matching

How articulation axioms are generated and how different schemas are integrated depends crucially on the process of matching them. For resource integration, the problem of matching is: given a representation for a concept in a local context, find its corresponding concept in the global context. There are several factors that affect this phase: there may be a mismatch between the local and global contexts in the depth of knowledge representing a concept, and there may be mismatches between

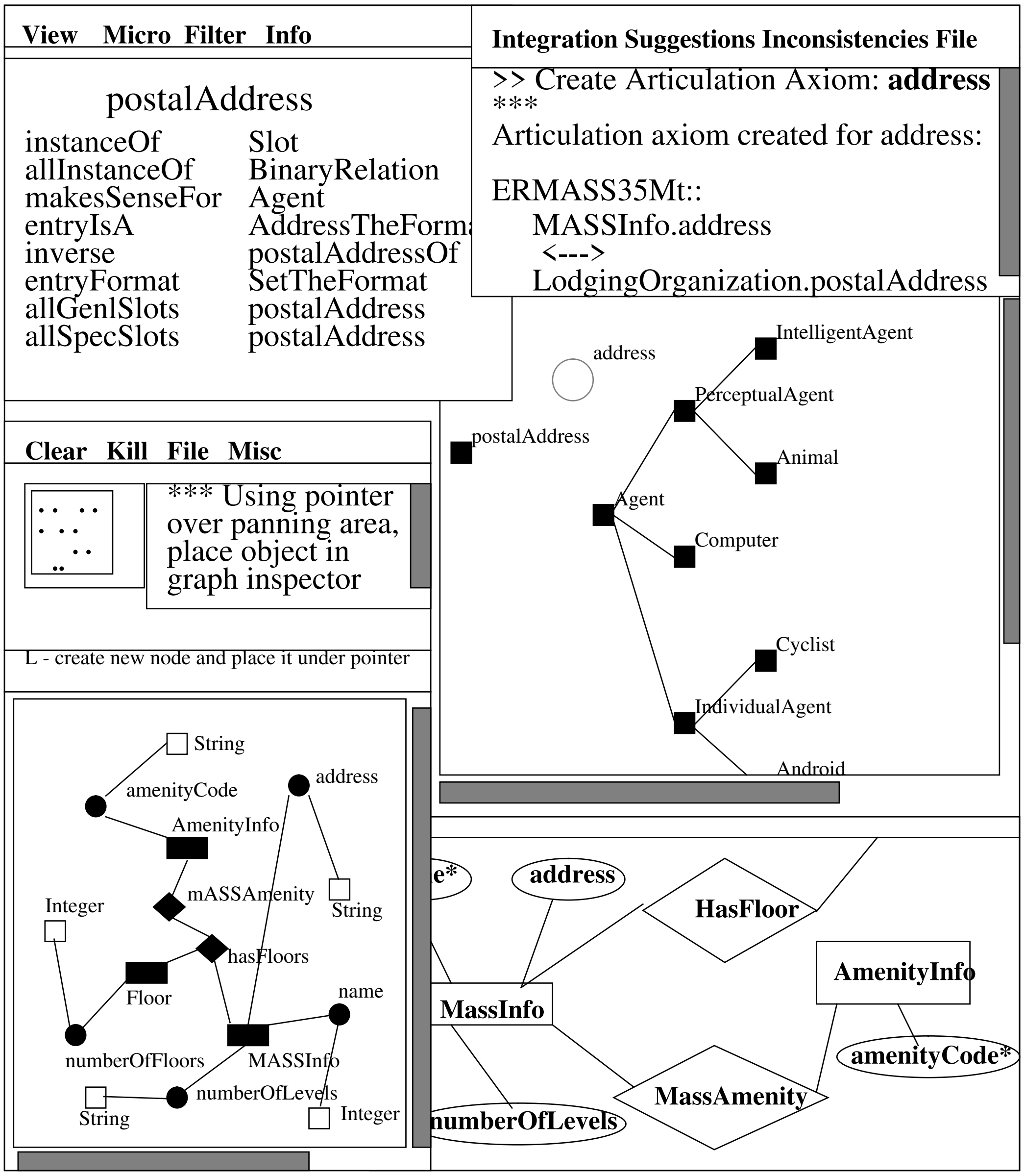


Figure 2.4. Sample MIST Screen

the structures used to encode the knowledge. For example, a concept in Cyc can be represented as either a collection or an attribute [LENA90].

If the global context's knowledge is more than or equivalent to that of the local context's for some concept, then the interactive matching process described in this section will find the relevant portion of the global context's knowledge. If the global context has less knowledge than the local context, then knowledge will be added to the global context until its knowledge equals or exceeds that in the local context; otherwise, the global context would be unable to model the semantics of the resource. The added knowledge refines the global context. This does not affect previously integrated resources, but can be useful when further resources are integrated.

Finding correspondences between concepts in the local and global contexts is a subgraph-matching problem. We base subgraph matching on a simple string matching between the names or synonyms of frames representing the model and the names or synonyms of frames in the global context. Matching begins by finding associations between attribute/link definitions and existing slots in the global context. After a few matches have been identified, either by exact string matches or by a user indicating the correct match out of a set of candidate matches, possible matches for the remaining model concepts are greatly constrained. Conversely, after integrating an entity or object, possible matches for its attributes are constrained.

Constructing Articulation Axioms

An articulation axiom is constructed for each match found. For example, the match between a relational attribute phone in model AAA and the Cyc slot phoneNumber yields the axiom

ist(Cyc phoneNumber(?L ?N)) $\Leftrightarrow$ ist(AAA phone(?L ?N))

which means that the phone attribute definition determines the phoneNumber slot in the global schema, and vice versa. Articulation axioms are generated automatically by instantiating stored templates with the matches found. These articulation axioms (in a simplified format) are stored in the Carnot global dictionary where they are accessible to the Carnot Distributed Semantic Query and Transaction Manager (see Section 2.5.2).

2.5 Information Management Infrastructure—ESS and DSQTM

The runtime Information Management Infrastructure is shown at the top of Figure 2.2. Client applications are shown on the left-hand side of the figure. These applications access database services through the Extensible Services Switch (ESS). Applications written in the C language use function calls to an implementation of the SQL Access Group Call Level Interface (SAG CLI) to request database access. The middle portion of Figure 2.2 is the Carnot implementation of middleware that connects client applications with the servers on the right side of the figure. A client application can connect to an ESS through the SAG CLI and access a specific re-

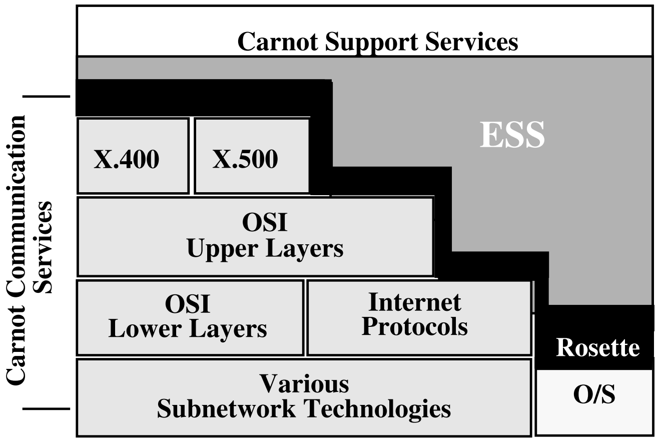


Figure 2.5. Carnot Communication and Support Layers

mote database through either the Remote Database Access protocol or through the ESS TreeSpace protocol. This simple type of access does not require any of the results of the enterprise modeling that were discussed in the previous section.

If a client application wishes to have a more abstract view of the information space, it can connect to the DSQTM ESS shown at the bottom of the center of Figure 2.2. The DSQTM uses the global dictionary, local schemas, articulation axioms, and declarative resource constraint base to translate an abstract request into a set of queries against a set of physical databases. The following sections will describe the ESS and the DSQTM in more detail.

2.5.1 Extensible Services Switch (ESS)

The Carnot Extensible Services Switch (ESS) provides interpretive access to communication resources, information resources and applications resident at a site in a distributed system. There are two sorts of paradigms that may profitably be used to understand the ESS. First, the ESS can be thought of as a component of a distributed command and control interpreter that is used to implement heterogeneous distributed transaction execution and generalized workflow control. Second, the ESS may be viewed as a programmable application layer communication front-end for applications and other resources within a distributed information system. The ESS is essentially a programmable glue for binding software components to one another in a manner that enhances interoperability. The coarse structure of the communication and support service layers is summarized in Figure 2.5.

Rosette Implementation of the Actor Model

The ESS is constructed on top of a high-performance implementation of an interpreter for the Actor model [AGHA86], [AGHA90], enhanced with object-oriented mechanisms for inheritance and reflection. The language of the interpreter is called Rosette [TOML92] and has been developed over a period of five years for both research in parallel algorithm development and interpretive control of distributed applications. Rosette is a prototype-based, object-oriented command language based on the Actor model. The syntax of the Rosette language is similar to Lisp or Scheme and the sequential aspects of the language model may be usefully compared with Self [UNGA87].

The Rosette interpreter is harmonized to the specific operating system facilities of a variety of platforms, and enhanced with a variety of distributed processing and communications facilities such as remote evaluation, tree spaces, and virtual synchrony. The workstation platforms on which Rosette is supported include: Sun3, Sun4, DEC5100, Hewlett Packard 700 and 800, IBM RS6000 running AIX, Silicon Graphics IRIS, and various Intel 386/486 based systems running UNIX System V Release 3 or 4 or 4.3 BSD Unix. The virtual machine, byte-code compiler, and a variety of primitive actors are implemented in C++.

There are two features of the Actor model that make it particularly well suited as the basis for an integration tool such as the ESS:

1. the basic semantics of the Actor model of computation is asynchronous communication among actors, which concurrently executing entities.
2. the Actor model includes a simple and powerful model for synchronizing and controlling the interference among concurrently executing threads of control.

In the Actor model everything is executed concurrently by default. It is necessary to explicitly indicate where there are data or control flow dependencies among steps. In Rosette this leads to ultra-lightweight threads that can be used:

1. to express the concurrent execution of components of a distributed job in a simple manner, and
2. to multiplex many independent activities through a single instance of the interpreter at a site.

The latter is important, because it permits more effective control of the processor resources at a site than traditional approaches using shell programs, where a new operating system process is instantiated for each nested activity that is expressed in the command language. This can become a significant drain on processor resources at a site. Rosette can perform multiple independent activities on behalf of arbitrary clients. Further, Rosette may be used to multiplex access to separate information resources under the control of a single instance of an ESS. This allows for sophisticated control of multiple resources to be expressed simply in the command language. Essentially, control at this level is used to effect policies of one sort or another concerning how resources are to be used in the enterprise.

Rosette Integration Facilities

Rosette incorporates several integration facilities that are particularly important for the interoperation of independent applications and resource managers. These include:

1. the dynamic definition of foreign language storage structures—this facility permits simple syntactic transformations of C language header files to be directly loaded into a Rosette environment so as to define various application or resource manager specific types as actors with the same underlying storage structure as that defined in C. Methods may be attached to these actors so that the structures may be used just like any other class of actors;

2. type-checked access to procedures generated by foreign language processors linked statically or dynamically into the runtime environment—this facility includes features for converting between types in Rosette, such as strings and numbers and those required or returned by the foreign language procedure, as well as a capability to clone returned static structures automatically. This facility uses a simple syntactic transformation of the ANSI C function prototype;

3. management of external events among applications—this facility virtualizes interrupt or signal handling among various threads executing in a single ESS environment and significantly reduces overhead due to polled interaction among such components as a communication package and a windowing package. Events are materialized as messages to actors representing one or another of the integrated resources within an ESS.

These basic integration facilities have been used, for example, to embed the following services in the ESS environment:

1. Carnot's Graphical Interaction Environment, which integrates X window, Motif widgets, MCC's 2D modeling system(Germ), and MCC's 3D modeling and animation system(Mirage) components. These components enable various interface builders to be layered in the ESS environment and can be used to develop "groupware" as well as conventional graphical interfaces.

2. Interfaces to Ingres, Sybase, and Oracle relational database systems and the Itasca and Objectivity object-oriented database systems. These interfaces provide query and transaction access to various database resources at a network site. Multiple resources may be supported via a single ESS, which permits an ESS to perform as a distributed transaction manager.

3. a distributed query and transaction planner based on the BellCore OMNIBASE query optimizer. This component is used to derive workflow scripts that perform distributed query and transaction processing.

4. Interfaces to ISODE 7.0/8.0, SunLink OSI, SunLink 3270, and DECNet OSI communications components. These services support the integration of applications within a sophisticated distributed environment without having to rework the applications themselves in order for them to participate in the environment.

5. SQL Access Group Remote Database Access(RDA) clients and servers. This service permits components connected to or embedded within an ESS to access database resources via an industry standard application protocol supported by a variety of independent vendors.

ESS Support for Distributed Processing

The above services are supported within the ESS environment by a variety of communications resource and distributed processing abstractions. These include:

1. ByteStream,
2. RosetteStream,
3. Remote evaluation,
4. TreeSpaces,
5. Virtual Synchrony, and
6. OSI

The ByteStream provides access to an uninterpreted stream of bytes, while the RosetteStream provides access to a stream of Rosette expressions. This latter provides basic support for remote evaluation and is used by several of the client applications of the ESS. The TreeSpace abstraction is a collection of classes built on RosetteStreams and provides functions similar to those of Linda's tuple spaces [CARR89] or ActorSpaces [AGHA93]. TreeSpaces are used extensively by the "distribution services" layer to manage the distribution of work and include a GroupSpace abstraction that provides a multicast facility obeying a form of virtual synchrony [BIRM91].

The OSI interfaces are illustrated in Figure 2.6. The interface to Association Control(ACSE) permits an ESS to act as both an initiator and a responder for OSI application layer associations. The interface to the Remote Operations Service(ROSE) permits an ESS to act as both an invoker and a performer of remote operations. The ESS provides support for both the ISO/CCITT Abstract Syntax Notation(ASN.1) and the Remote Operations Notation (RON). These capabilities are particularly important as they allow an application designer to specify an application completely in terms of the formal description of the application protocol, rather than having to mentally translate into some implementation language representation of the corresponding protocol data units.

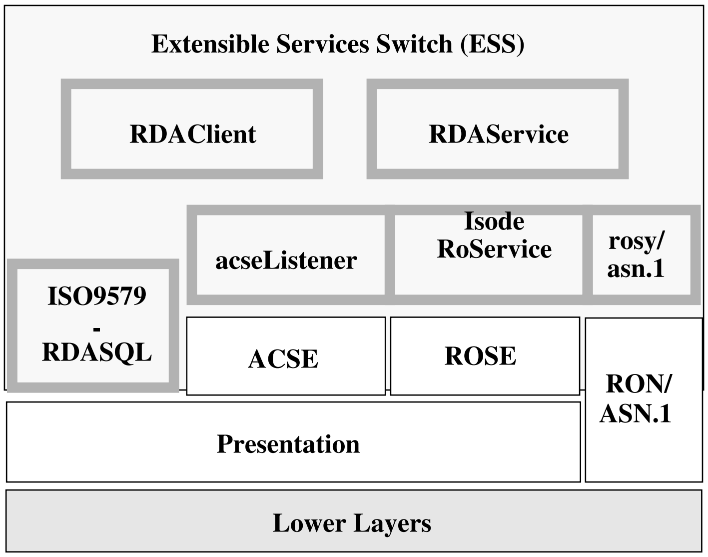


Figure 2.6. OSI Application Layer Interfaces in the ESS

The RDA components support the SQL Access Group definition of the Remote Database Access protocol (FAP-92). In an application, there may be a variety of database resources accessible locally through an ESS and in this case the 'application entity qualifier' is used to select the appropriate advertised database service at an ESS. The RDA facility has been used in a variety of application contexts including providing access to chemical property databases by advanced logic data language and natural language processors. This example is typical of the off-loading of communications from a variety of applications so that there is a net reduction in maintenance and an increase in flexibility.

The principal components of this facility are illustrated in Figure 2.7. The RDA-Client actor implements a translation from a common remote evaluation protocol to the RDA protocol. The remote evaluation protocol is used by a client of the ESS to access SQL resources within the system. In this case, the client is a Graphical Interface Environment (GIE) developed as part of the Carnot project. The ISO9579-I/F implements the RDA protocol machine, seeing to it that application protocol data units are properly encoded and decoded and that state transitions are legal. This agent was handcrafted for FAP-92, since that protocol is specified without benefit of the Remote Operations Notation.

The RDAService actor implements a translation from the RDA protocol to an SQL remote evaluation protocol. The Itasca actor implements a translation from

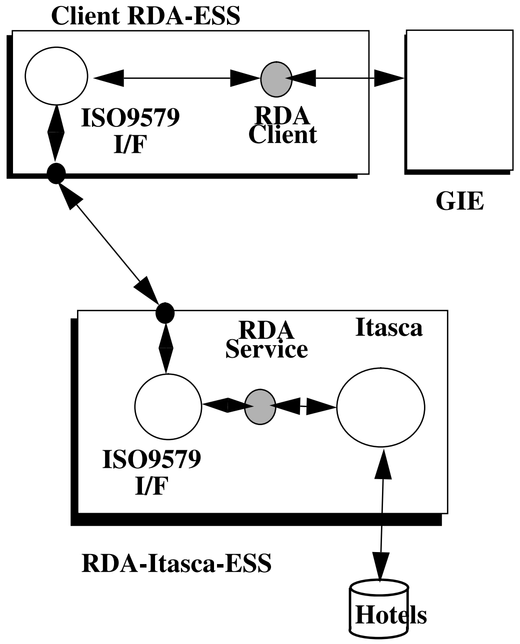


Figure 2.7. Principal actors implementing RDA support in the ESS

the SQL remote evaluation protocol to a local protocol for accessing the Itasca object-oriented database system. Other database systems for which interfaces have been developed include Ingres, Oracle, Objectivity, and Sybase. The SQL remote evaluation protocol is very simple to implement and well suited for attaching a variety of clients to the ESS environment. The protocol serves a purpose quite similar to an API, but for external processes.

2.5.2 Distributed Semantic Query and Transaction Manager

The Distributed Semantic Query and Transaction Manager is our initial implementation to provide query and relaxed transaction processing services. It dynamically expands a query to include access to all semantically equivalent and relevant information resources, and also groups any updates to these resources as a set of independent transactions that interact according to a dynamically defined relaxed transaction semantics [WOEL93a]. DSQTM is based on the OMNIBASE multidatabase system [RUSI89]. Figure 2.8 is a block diagram of the DSQTM. The query graph generator module accepts SQL and generates a query graph using information from a global dictionary. The query graph is annotated with physical information resource locations that were either provided in the initial SQL query

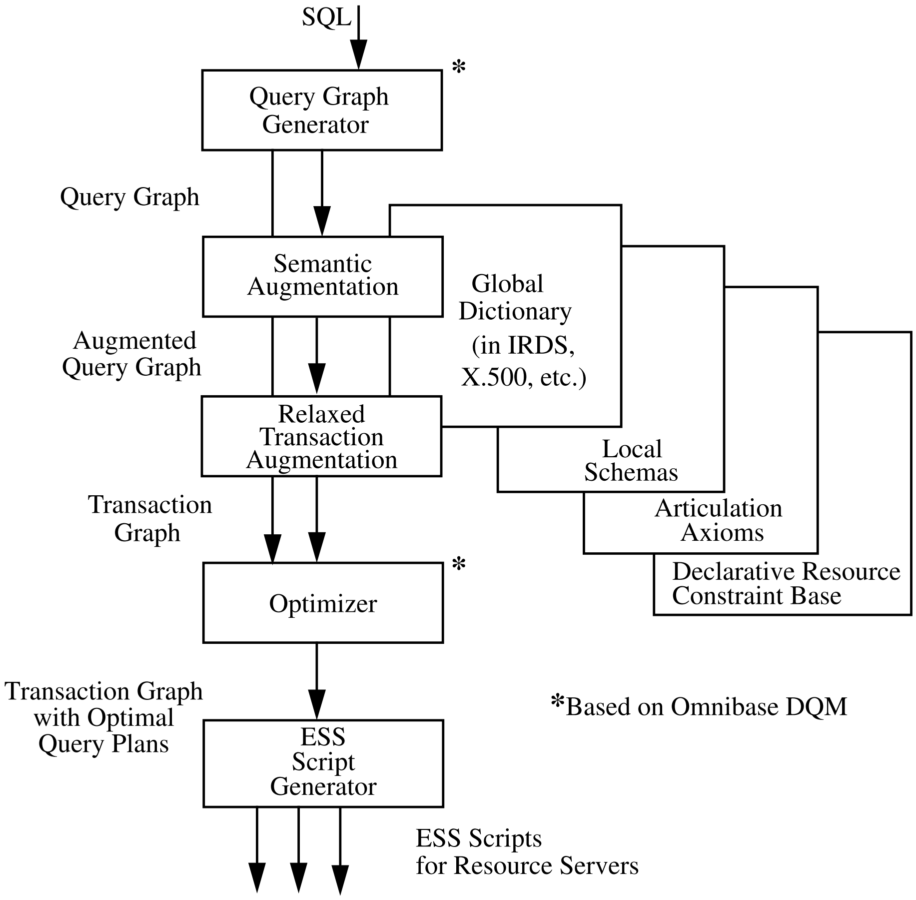


Figure 2.8. Block Diagram of DSQTM

or that were found in the global dictionary. The query graph is then optionally passed to the semantic augmentation module, which uses articulation axioms to expand the query graph to include nodes representing other information resources that contain relevant information. Articulation axioms are generated by the MIST tool described in Section 2.4.

Next, if the original query requested modifications to an information resource, the query graph is passed to the relaxed transaction augmentation module, which uses rules in the declarative resource constraint base to determine update transactions for other information resources that will be necessary to maintain consistency. The format of the rules is similar to those in [SHET92]. Separate query graphs are built for each of these transactions. The declarative resource constraint base also specifies the relaxed transaction model that will govern the interaction among the transactions. The model is synthesized from dependencies declared among the transactions in a language similar to ACTA [CHRY90]. The separate query graphs are then related to each other within a transaction graph.

The transaction graph is passed to the optimizer module, which generates an optimal query plan for each query graph node in the transaction graph. The query plan specifies the ordering of joins and the flow of intermediate results among the remote systems. The query processing strategy is based on a dataflow execution model similar to the strategy for distributed query processing implemented for the ORION distributed object-oriented database system [JENQ90]. The query plan(s) and transaction graph are passed to the ESS script generator, which builds a script to be executed at each site that will participate in the execution of the transactions.The script controls the execution of subqueries or subtransactions at a site, controls the flow of data to and from the site, does necessary data value mapping between databases, and controls the transaction semantics [TOML93]. The script is executed under the control of the ESS distributed execution environment described in Section 2.5.1. Communication with an individual database server is via either TCP, RDA, or ESS tree space.

DSQTM has been implemented as an ESS actor object. The DSQTM actor responds to a set of messages that include the typical functions provided by a database server (connect to database, begin transaction, execute query, etc.). Much of the functionality of DSQTM is provided by C and C++ functions, which are called through the ESS foreign function interface. This allows ESS to control concurrent execution of functions while implementing individual functions in C and C++.

The following sections describe semantic augmentation and relaxed transaction augmentation modules in more detail.

2.5.3 Semantic Augmentation of Query Graphs

Using the articulation axioms, the semantic augmentation module modifies and expands the query graph. This is similar to expanding a relational query on a view into an equivalent query on a set of base relations. However, additional transformations must also be made. For example, a predicate in one relation describing

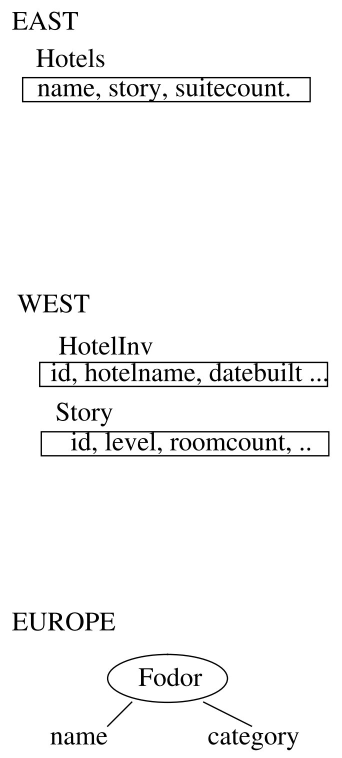


Figure 2.9. Example databases

hotels might be expressed as "rate $>$ 200" and in another relation as "category = expensive". Furthermore, the "rate" value of 200 may have to be changed to "expensive" before returning it to the user.

Articulation axioms are grouped to define "semantic closures." A query associated with a semantic closure will be expanded using only the articulation axioms that are in the semantic closure. Thus, by specifying a proper set of articulation axioms, users can control whether their query will be expanded or shrunk, and how far the expansion/shrinkage will go.

To illustrate the semantic augmentation module, let us consider three travel databases, EAST, WEST, and EUROPE, shown in Figure 2.9.

We assume these databases are integrated through a knowledge base containing knowledge about lodging organizations. The concept of "hotel" is represented as a table "Hotels" in EAST, a table "HotelInv" (Hotel Inventory) in WEST, and a class "Fodor" in EUROPE. The integration of these different representations is

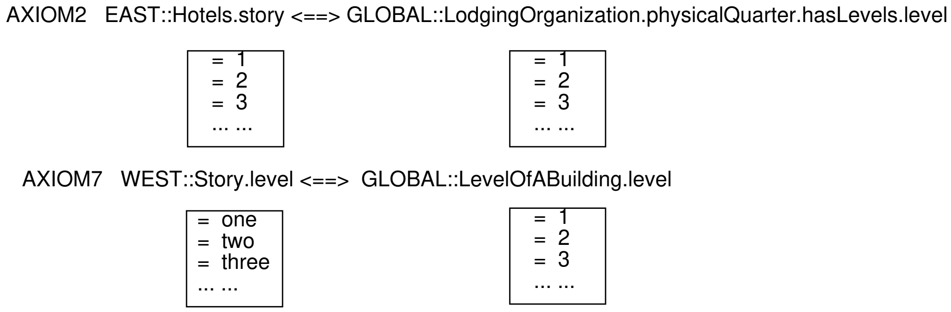


Figure 2.10. Axioms for Example Databases

accomplished by a set of articulation axioms that includes the following:

AXIOM1 EAST::Hotels.name $\Leftrightarrow$ GLOBAL::LodgingOrganization.name
AXIOM2 EAST::Hotels.story $\Leftrightarrow$
GLOBAL::LodgingOrganization.physicalQuarter.hasLevels.level
AXIOM3 EAST::Hotels.suitecount $\Leftrightarrow$
GLOBAL::LodgingOrganization.physicalQuarter.hasLevels.rooms
AXIOM6 WEST::HotelInv.hotelname $\Leftrightarrow$ GLOBAL::LodgingOrganization.name
AXIOM7 WEST::Story.level $\Leftrightarrow$ GLOBAL::LevelOfABuilding.level
AXIOM8 WEST::HotelInv.id $\bowtie$ Story.id $\Leftrightarrow$
GLOBAL::LodgingOrganization.physicalQuarter.hasLevels
AXIOM9 WEST::Story.roomcount $\Leftrightarrow$ GLOBAL::LevelOfABuilding.rooms
AXIOM11 EUROPE::Fodor.name $\Leftrightarrow$ GLOBAL::LodgingOrganization.name

Articulation axioms can be used in a number of ways to translate from concepts in one database to concepts in other databases. The simplest way is a concept-to-concept mapping. For example, Fodor.name is matched to LodgingOrganization.name, which is in turn matched to Hotels.name in the EAST database. The match can also be concept-to-chain. For example, Hotels.story is mapped to LodgingOrganization.physicalQuarter.hasLevels.level, which takes a chain of two axioms (LodgingOrganization.physicalQuarter.hasLevels from AXIOM8 and LevelOfABuilding.level from AXIOM7) to be translated into HotelInv(joined)Story.level. Furthermore, the mappings can be fork-like. In AXIOM8, for example, the two relations HotelInv and Story are joined to match to the global concept LodgingOrganization.physicalQuarter.hasLevels. Finally, articulation axioms can be associated with value mappings, which specify how the values in one local concept should be translated into the values of concepts in other local databases. For example, suppose that the values of Story.level in WEST are "one", "two", "three", etc., the values of Hotels.story are 1, 2, 3, etc., and the values of LevelOfABuilding.level are 1, 2, 3, etc., then the value mappings associated with the relevant axioms are as

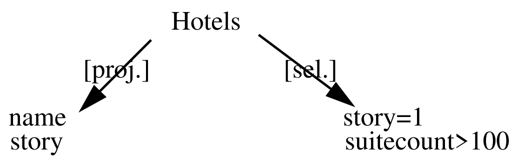


Figure 2.11. Query Graph for Example Databases

shown in Figure 2.10.

With these value mappings, Hotels.story=1 in the EAST database can be translated to Story.level="one" in the WEST database. In general, each set of value mappings is represented as a table that maps (operation$_i$ value$_i$) to (operation$_j$ value$_j$).

The semantic augmentation module accepts a query graph that is generated by parsing the original query. A query graph is a tree of nodes that represent relations and that specify how these relations are tailored (by projections on columns and selections on rows) before they are joined through some expressions.

Given a query graph, the semantic augmentation module searches through the projections and selections of each existing relation node, and checks if it can be expanded, through the articulation axioms in the current semantic closure, to new projections and selections on new relations. A projection or a selection is expandable if it matches to some articulation axioms in the current closure. The other "end" of matching are the projections or selections to be created.

Consider, for example, the following SQL query against the EAST database:

SELECT name, story FROM Hotels WHERE story=1 AND suitecount >100.

The query graph generated from this query is as shown in Figure 2.11.

This graph is expanded into the graph shown in Figure 2.12 by the semantic augmentation module based on the axioms given above.

HotelInv.hotelname is created because of the mappings (AXIOM1 and AXIOM6) from Hotels.name in the original graph. The join between HotelInv and Story is created because of the mapping (AXIOM2, AXIOM7 and AXIOM8) triggered by Hotels.story in the original graph. The value mappings associated with this path also translate story=1 to level = "one". Finally, the selection Story.roomcount > 100 is created through AXIOM3 and AXIOM9 from the original selection Hotels.suitecount > 100. (In this case, there is no value mapping associated with the path, so "> 100" is translated to "> 100" by default.)

Relaxed Transaction Augmentation

The relaxed transaction augmentation module generates a transaction graph that may include updates to additional information resources and constraints on the execution of these updates [WOEL93b]. For example, Figure 2.13 represents a task graph that has been created when the deletion of a booking is requested in a travel

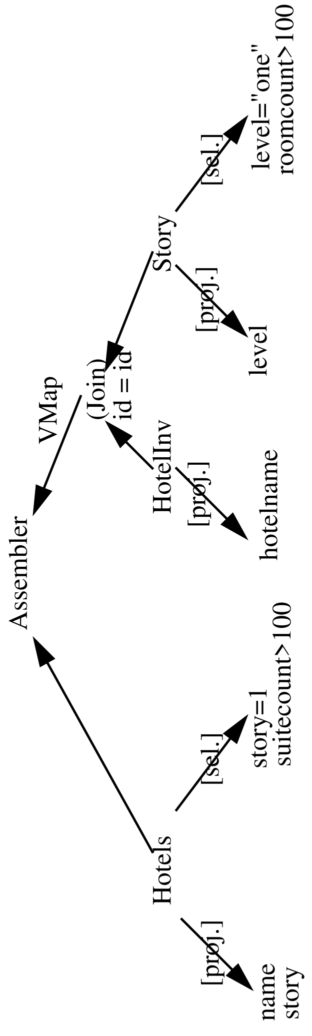


Figure 2.12. Augmented Query Graph for Example Databases

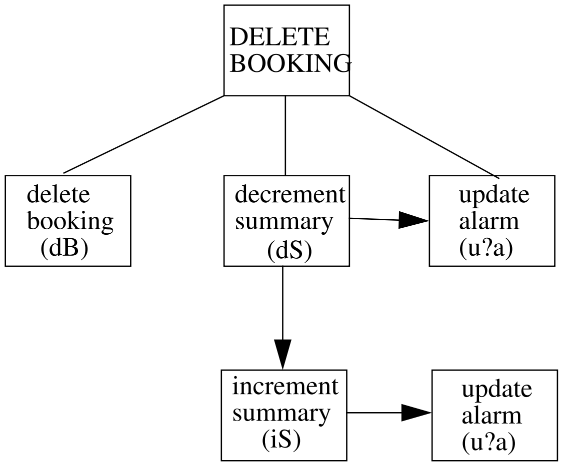


Figure 2.13. Task Graph for the Delete Booking Example

database. Assuming that there is a booking relation in one database that represents the bookings for each travel agent and that there is a summary relation in another database that keeps a count of the number of bookings per agent, then, when a booking is deleted, the booking count for the agent should also be decremented. 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 2.2 shows the tasks involved in this example; where dB, dS, iS, and u?a denote the delete-booking, decrement-summary, increment-summary, and update-alarm tasks, respectively.

There are dependencies among the task boxes in Figure 2.13. 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 that 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 that enforces these dependencies. We have chosen the third strategy, described further below.

The specification and enforcement of intertask dependencies has recently received considerable attention [CHRY90, ELMA90, ELMA92, KLEI91]. Following [CHRY90] and [KLEI91], 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 [KLEI91]:

1. $e_1 \rightarrow e_2$: 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 e_1 must precede e_2.

Examples of execution dependencies defined in the literature include:

1. Commit Dependency [CHRY90]: 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 [CHRY90]: 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 $ab_B \rightarrow ab_A$.

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 2.14 shows an example task state transition diagram taken from [KLEI91]. 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 2.15. The intertask dependencies are shown as "links" between significant events of various tasks. For example, the dependency $(ab(dB) < dn(dS)) \rightarrow ab(dS)$

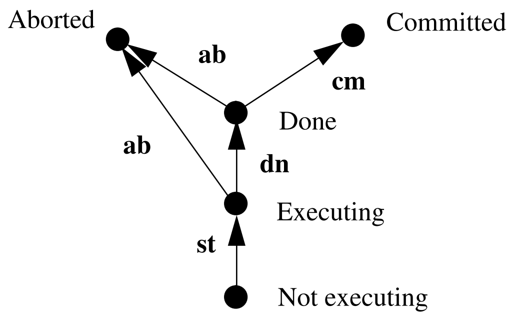


Figure 2.14. An Example Task State Transition Diagram

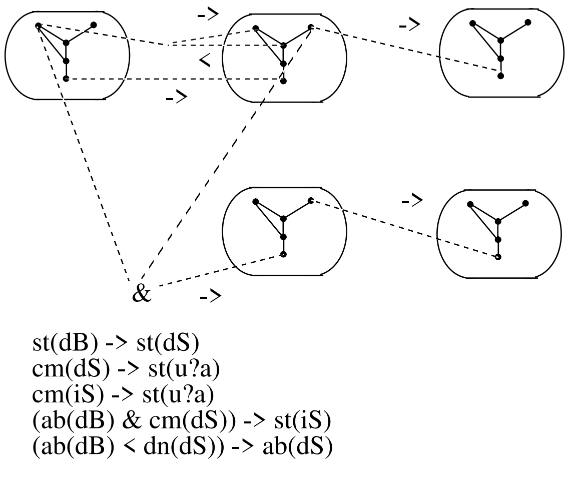


Figure 2.15. Dependencies between Significant Events

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 Computation Tree Logic [EMER82] and the implementation of a scheduler to enforce these dependencies is described in [ATTI93]. The scheduler is implemented in the concurrent actor language Rosette. We are continuing to enhance this approach by exploiting results in action logics [PRAT90] and to experiment with the construction of advanced transaction models.

2.6 Building Applications Using Carnot

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.

A demonstration application has been developed at MCC that focuses on interoperability among database servers. MCC has made a strong commitment to comply with industry standards and to actively influence industry standards to insure that standards will not hinder future advances in technology. As part of this commitment, Carnot has participated in the SQL Access Group, 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. The SQL Access Group is attempting to accelerate the acceptance of the ISO Remote Database Access (RDA) protocol. In July 1991 a group of 15 SQL Access Group members, including MCC, demonstrated interoperability among their various hardware and software systems using the RDA protocol. Figure 2.16 describes the vendors that participated in the demonstration.

The MCC database server was an Itasca object-oriented database system with an Object SQL interface, a subset of which provides standard SQL functionality. The Object SQL was written by the Carnot project as 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) and the Carnot DSQTM.

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 GIE, which allowed the user to click on an icon to generate a query.The query was decomposed into separate subqueries, one for each database server. Two or three database servers were usually accessed. The subqueries were sent to the database servers using the RDA protocol. The subqueries were executed

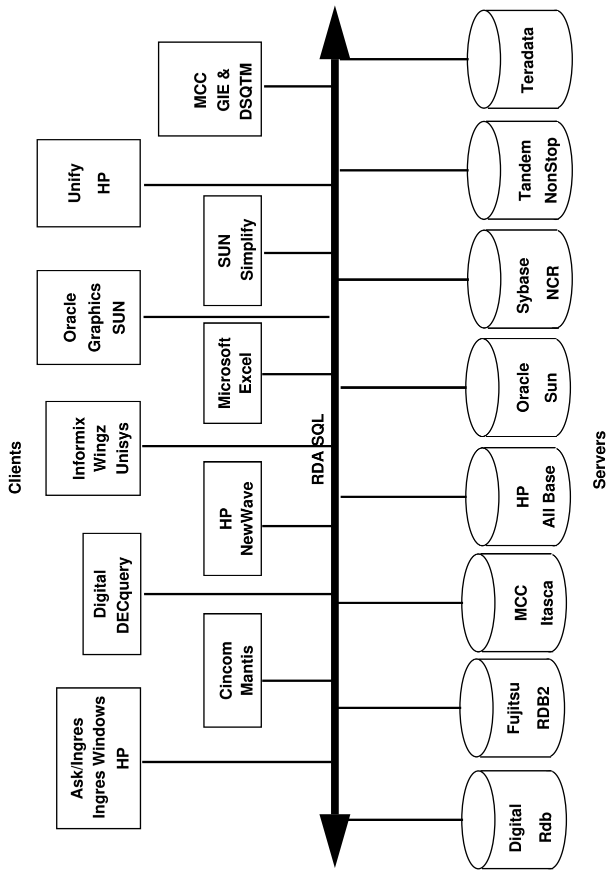


Figure 2.16. SQL Access Group Demonstration

concurrently, returning their results to the DSQTM, which assembled the results into a single answer. The GIE then created instances of Itasca classes to represent the results. These instances were presented to the user as icons, which could be further manipulated.

Since the SQL Access Group demo, the Carnot group has enhanced the prototype environment. We have developed RDA, tree space, and TCP servers using Ingres and Oracle DBMS's. There are now two GIE applications, a Manager Application and a Travel Agent Application. The Manager Application enables the manager in a company to make travel plans directly or to review the company's travel arrangements. The Travel Agent Application provides similar capabilities for the travel agency and, additionally, provides active icons for monitoring the status of travel arrangements by many companies.

In addition to the travel database, a Fodor database containing general information on hotels has been added. A query for information about hotels in the travel database can now be expanded by the DSQTM semantic augmentation module to also produce a subquery to the Fodor database for additional information. The results are merged to form an Itasca instance that can be displayed and manipulated.

A Carnot knowledge-based Distributed Communicating Agent (DCA) is used to implement the declarative resource constraint base. The delete booking example described in Section 5.2.2 has been implemented as part of this demo.

2.7 Acknowledgments

The following individuals at MCC have also contributed to the design and development of the Carnot prototype: Natraj Arni, Paul Attie, Bharat Mitbander, Steven Tighe, Nicki Turman, C. Unnikrishnan, and Dallas Webster.

Bibliography

[AGHA86] Agha, G. Actors. The MIT Press, Cambridge, MA, 1986.

[AGHA90] Agha, G. "Concurrent Object-Oriented Programming." Communication of the ACM, September 1990, pp. 125–141.

[AGHA93] Agha, G. and C. Callsen. "ActorSpace: An Open Distributed Programming Paradigm." in Proceedings Principles and Practice of Parallel Programming 1993.

[AHLS90] Ahlsen, M. and P. Johannesson, "Contracts in Database Federations," in S. M. Deen, ed., Cooperating Knowledge Based Systems 1990, Springer-Verlag, London, 1991, pp. 293–310.

[ATTI93] Attie, P., M. Singh, A. Sheth, and M. Rusinkiewicz. "Specifying and Enforcing Intertask Dependencies". Proceedings of the 19th International Conference on Very Large Data Bases (VLDB), Dublin, Ireland, August 1993.

[BIGE93] Bigerstaff, T., Mitbander, B. and D. Webster. "The Concept Assignment Problem in Program Understanding", 15th Intl Conf. on SWE (ICSE), May 17–21, 1993, pp. 482–498.

[BIRM91] Birman, K., A. Schiper, and P. Stephenson. "Lightweight Causal and Atomic Group Multicast." ACM Transactions on Computer Systems, 9(3):272–314, August 1991.

[BUNE90] Buneman, P., S. B. Davidson, and A. Watters, "Querying Independent Databases," Information Sciences, Vol. 52, Dec. 1990, pp. 1–34.

[CARR89] Carriero, N. and D. Gelernter. "Linda in Context." CACM, Volume 32, number 4, 1989.

[CHRY90] Chrysanthis, P. and K. Ramamritham. "ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior", Proc.of ACM SIGMOD, p. 194–203, 1990.

[ELMA90] Elmagarmid, A., Y. Leu, W. Litwin, and M. Rusinkiewicz. "A Multidatabase Transaction Model for Interbase". Proceedings of the VLDB Conference, August, 1990.

[ELMA92] Elmagarmid, A., editor. Database Transaction Models for Advanced Applications, Morgan Kaufmann, 1992.

[EMER82] Emerson, A. and E. Clarke. "Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons". Science of Computer Programming, vol.2, 1982, 241–266.

[GUHA90] Guha, R.V. "Micro-theories and Contexts in Cyc Part I: Basic Issues," MCC Technical Report Number ACT-CYC-129-90, Microelectronics and Computer Technology Corporation, Austin, TX, June 1990.

[HEIM85] Heimbigner, D. and D. McLeod, "A Federated Architecture for Information Management," ACM Transactions on Office Information Systems, Vol. 3, No. 3, July 1985, pp. 253–278.

[HUHN92] Huhns, M., N. Jacobs, T. Ksiezyk, W.M. Shen, M. Singh, and P. Cannata, 1992."Enterprise Information Modeling and Model Integration in Carnot", in Charles J. Petrie Jr., ed., Enterprise Integration Modeling: Proceedings of the First International Conference, MIT Press, Cambridge, MA, 1992.

[JENQ90] Jenq, P., D. Woelk, W. Kim, and W. Lee. "Query Processing in Distributed ORION", International Conference on Extending Database Technology, p. 169–187, March, 1990.

[KLEI91] Klein, J. "Advanced Rule Driven Transaction Management." Proceedings of the IEEE COMPCON, 1991.

[LENA90] Lenat, D. and R. V. Guha, Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project, Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.

[LITW90] Litwin, W., L. Mark, and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases," ACM Computing Surveys, Vol. 22, No. 3, Sept. 1990, pp. 267–296.

[PRAT90] 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.

[RUSI89] Rusinkiewicz, M., Elmasri, R., Czejdo, B., Georakopoulous, D., Karabatis, G., Jamoussi, A., Loa, L. and Li, Y. 1989. "OMNIBASE: Design and Implementation of a Multidatabase System". Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing, pp. 162–169.

[SHET92] Sheth, A., M. Rusinkiewicz, and G. Karabatis, "Using Polytransactions to Manage Interdependent Data," in Database Transaction Models for Advanced Applications, edited by A. Elmagarmid, Morgan Kaufmann, 1992.

[TOML92] Tomlinson, C., G. Lavender, G. Meredith, D. Woelk, and P. Cannata. "The Carnot Extensible Services Switch (ESS) - Support for Service Execution," in Charles J. Petrie Jr., ed., Enterprise Integration Modeling: Proceedings of the First International Conference, MIT Press, Cambridge, MA, 1992.

[TOML93] Tomlinson, C., P. Attie, P. Cannata, A. Sheth, M. Singh, and D. Woelk, "Workflow Support in Carnot", Data Engineering Bulletin, Vol. 16, No. 2, June, 1993, pp. 33–36.

[UNGA87] Ungar, D. and R. B. Smith. "Self: The Power of Simplicity." in ACM OOPSLA Proceedings, 1987.

[WOEL92] Woelk, D., W. Shen, M. Huhns, and P. Cannata, "Model Driven Enterprise Information Management in Carnot", in Charles J. Petrie Jr., ed.,. Enterprise Integration Modeling: Proceedings of the First International Conference, MIT Press, Cambridge, MA, 1992.

[WOEL93a] 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.

[WOEL93b] Woelk, D., P. Attie, P. Cannata, G. Meredith, A. Sheth, M. Singh and C. Tomlinson. "Task Scheduling Using Intertask Dependencies in Carnot", ACM SIGMOD Proceedings, 1993, pp. 491–498.