Information Management for Cooperative Engineering

Michael N. Huhns

Department of Electrical and Computer Engineering
University of South Carolina
Columbia, SC 29208, USA
huhns@sc.edu

Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534, USA
singh@ncsu.edu

Abstract

In this paper we describe how a set of autonomous computational agents can cooperate in providing coherent management of information in environments where there are many diverse information resources. The agents use models of themselves and of the resources that are local to them. Resource models may be the schemas of databases, frame systems of knowledge bases, or process models of business operations. Models enable the agents and resources to use the appropriate semantics when they interoperate. This is accomplished by specifying the semantics in terms of a common ontology. We discuss the contents of the models, where they come from, and how the agents acquire them. We then describe a set of agents for telecommunication service provisioning and show how the agents use such models to cooperate. Their interactions produce an implementation of relaxed transaction processing.

1 Introduction

World-wide production of manufactured goods is currently being affected by six related factors:

- There are pressures for a shorter time-to-market, forcing a need for all aspects of product engineering—from conceptualization through delivery and maintenance—to be considered simultaneously.

- There are changes in the artifacts of production, in that many products that used to be standardized are being specially designed for each customer, and more complicated products, such as space stations and fusion power plants, are being attempted.

- There are increasing data, knowledge, and experience being accumulated about all aspects of production processes, which can be used to aid future production processes.

- There are now a plethora of tools for aiding product engineering, including tools for simulation, visualization, layout, test, aesthetics, compliance with standards, and manufacturability.

- The scope of the problem has increased to the point that teams of engineers are typically required.

- Engineering, as a part of overall business operations, can no longer be done in isolation, but must be done in a global context, i.e., as part of an enterprise. A characteristic of such enterprises is that their information systems are large and complex, and the information is in a variety of forms, locations, and computers. The topology of these systems is dynamic and their content is changing so rapidly that it is difficult for a user or an application program to obtain correct information, or for the enterprise to maintain consistent information.

The overall trend in each of the six factors has been towards increasing the complexity of the engineering task. This in turn has placed additional demands on the computational aids for engineering, with the foremost demands being for interoperability and coherent access to all the relevant information available.

Imagine the following scenario. An engineer designing the case for a new notebook computer must choose a fastener to attach two of the pieces. There are three alternatives—rivets, bolts, and adhesive—and the engineer's design system must provide him with comparative information about the sizes, strengths, reliabilities, costs, and availabilities of these three. There are two manufacturers of rivets and one each for bolts and adhesive. Information about the properties and costs of the fasteners are in databases accessible electronically. However, because the databases have been developed independently, their semantics are different. For example, one manufacturer prices rivets each, and the other by the hundred. There are also price breaks based on quantity. To take these into account, the design system must have access to sales projections for the notebook computer from marketing.

The engineer decides to use rivets, but the standard sizes listed in the database are not exactly right. Another engineer at the rivet company is contacted about designing a custom-sized rivet, but she has to query her manufacturing system to determine the date by when a sufficient quantity can be produced. The first engineer's design system uses this information to predict the availability of the notebook computer product. When a fastener order is finally placed, it results in transactions being committed on engineering, manufacturing, and sales information systems at several companies. As the product cycle progresses, changes might cause the information systems to be updated, with some of the transactions either rolled back or compensated.

There are four major techniques for coping with the size and complexity of enterprise information systems such as these: modularity, distribution, abstraction, and intelligence, i.e., being smarter about how you seek and modify information. The use of intelligent, distributed modules is one way of combining all four of these techniques, yielding a ***distributed artificial intelligence*** approach.

Distributed artificial intelligence (DAI) [Huhns 1987; Gasser and Huhns 1989; Huhns and Singh 1997] provides some of the technology needed for this integration and interaction. DAI is concerned with how a decentralized group of intelligent computational agents should coordinate their activities to achieve their goals. When pursuing common or overlapping goals, they should act cooperatively so as to accomplish more as a group than individually; when pursuing conflicting goals, they should compete intelligently. Interconnected agents can cooperate in solving problems, share expertise, work in parallel on common problems, be developed, implemented, and maintained modularly, be fault tolerant through redundancy, represent multiple viewpoints and the knowledge of multiple human experts, and be reusable.

In accord with this approach, we describe in this paper how to distribute and embed computational agents throughout an enterprise. The agents are knowledgeable about information resources that are local to them, and cooperate to provide global access to, and better management of, the information. For the practical reason that the systems are too large and dynamic (i.e., open) for global solutions to be formulated and implemented, the agents need to execute autonomously and be developed independently. To cooperate effectively, the agents must either *have models of each other and of the available information resources* or *provide models of themselves*. We focus on the latter in this paper.

For such an open information environment, the questions arise: what should be modeled, where do models come from, what are their constituents, and how should they be used? We discuss the types of models that might be available, and how agents can acquire them at compile time and use them at run time. We use the ontology developed for the large knowledge-based system, Cyc, for semantic grounding of the models. The ontology is made commonly available by providing access to it over a network, such as the Internet or one of its commercial variants. At compile time, semantic mappings are constructed relating schema-level models of each information system component to the common ontology. At run time, a distributed truth maintenance system [Huhns and Bridgeland 1991] is used to maintain semantic consistency of information. The truth maintenance system is executed by knowledge-based mediating agents—one for each user, resource, application, and interface—that actively monitor integrity constraints. In addition, the agents help locate and provide access to the most appropriate information for each user or application.

We then describe a set of agents for telecommunication service provisioning as an example of parameterized design. For these agents—a scheduling agent, a schedule-repairing agent, a schedule-processing agent, and an interface agent—we describe their models and how they use them to cooperate. We also describe the use of actors [Agha 1986]—one per agent—who manage their communications. Each actor independently maintains the relationship between its agent and the common ontology (in the form of the semantic mappings), and updates that relationship as the ontology changes or the agent evolves.

This achieves our research goal of providing interoperation among separately developed information resources, thereby enabling them to be accessed and modified coherently. Our system includes the computational infrastructure, from network communications to consistency maintenance, that cooperative engineering also requires.

2 Modeling

Enterprise information modeling is a corporate activity that produces the models needed for interoperability. The resultant models should describe all aspects of a business environment, including

- databases
- database applications
- software repositories
- part description repositories
- expert systems, knowledge bases, and computational agents
- work flows, and the information they create, use, maintain, and own, and
- the organization itself.

The models provide online documentation for the concepts they describe. They enable application code and data to be reused, data to be analyzed for consistency, databases to be constructed automatically, the impact of change on an enterprise to be assessed, and applications to be generated automatically.

An enterprise might have many models available, each describing a portion of the enterprise and each constructed independently. For example,

- the information present in a database is modeled by the schema for the database, which is produced through a process of logical data modeling
- the data values present in a database are modeled (weakly, in most cases) by data dictionary information, which is produced through data engineering
- the information present in an object-centered knowledge base is modeled by the ontology of the objects, which is produced through ontological engineering
- process models, possibly in the form of Petri nets or IDEFx descriptions, are produced through logical process modeling
- STEP (Standard for the Exchange of Product model data) schemas, written in Express, are produced from component and physical process modeling.

Although it might appear that interoperability would require all of these models to be merged into a single, homogeneous, global model, this is *not* the case in our approach. Instead, there are good reasons for retaining the many individual models: 1) they are easier to construct than a single large model; 2) enterprises may be formed dynamically through mergers, acquisitions, and strategic alliances, and the resultant enterprises might have inherited many existing models; 3) because enterprises are geographically dispersed, their resources are typically decentralized; and 4) as enterprises (and thus models) evolve, it is easier to maintain smaller models.

Unfortunately, the models are often mutually incompatible in syntax and semantics, not only due to the different things being modeled, but also due to mismatches in hardware, operating systems, data structures, and corporate usage. In attempting to model some portion of the real world, information models necessarily introduce simplifications and inaccuracies that result in semantic incompatibilities. However, the individual models must be related to each other and their incompatibilities resolved [Sheth and Larson 1990], because

- A coherent picture of the enterprise is needed to enable decision makers to operate the business efficiently and designers to evaluate information flows to and from their particular application.

- Applications need to interoperate correctly across a global enterprise. This is especially important due to the increasing prevalence of strategic business applications that require *intercorporate linkage*, e.g., linking buyers with suppliers, or *intracorporate integration*, e.g., producing composite information from engineering and manufacturing views of a product.

- Developers require integrity validation of new and updated models, which must be done in a global context.

- Developers want to detect and remove inconsistencies, not only among models, but also among the underlying business operations that are modeled.

We utilize a mediating mechanism based on an existing common ontology to yield the appearance and effect of semantic homogeneity among existing models. The mechanism provides logical connectivity among information resources via a semantic service layer that automates the maintenance of data integrity and provides an enterprise-wide view of all the information resources, thus enabling them to be used coherently. This logical layer is implemented as a network of interacting agents. Significantly, the individual systems retain their autonomy. This is a fundamental tenet of our Carnot architecture [Woelk *et al*. 1996; Singh *et al*. 1997].

3 Information Consistency

What technology is available to achieve the requisite consistency among different information resources?

Distributed truth maintenance: There are many desirable properties for a knowledge base, such as completeness, conciseness, accuracy, and efficiency. For an agent that can reason nonmonotonically, there are additional properties used to describe the *integrity* of the knowledge base: stability, well-foundedness, and logical consistency. In addition, an agent’s algorithm for maintaining well-founded stable states of its knowledge base should be *complete*, in that the algorithm should find a well-founded stable state if it exists. Each agent should have a complete algorithm for maintaining the integrity of its own knowledge base.

Truth maintenance systems are a common way to achieve knowledge base integrity in a single agent system, because they deal with the frame problem, they deal with atomicity, and they lead to efficient search. Furthermore, the justification networks they create can be used for nonmonotonic reasoning, problem-solving explanations to a user, explanation-based learning, and multiagent negotiation.

However, the above definitions of properties for a single knowledge base are insufficient to characterize the multiple knowledge bases in a multiagent environment. When agents that are nonmonotonic reasoners exchange beliefs and then make inferences based on the exchanged beliefs, then concepts of *distributed* knowledge-base integrity are needed [Huhns and Bridgeland 1991].

Nonmonotonic reasoning: Agents need to be able to maintain independent viewpoints and skepticism until they receive convincing evidence otherwise, but they should then be able to revise their beliefs consistently.

Negotiation: Negotiation has been explored as a means to mediate among conflicting agents. Existing systems involve either monotonic reasoners, or nonmonotonic, but memoryless, reasoners, i.e., reasoners that simply discard old solutions and re-solve in the face of conflicts. Negotiation is likely the correct approach, but the agents must be able to revise their plans incrementally as they interact. They must be able to communicate directly, with each other and with human agents, and they must be able to assess and maintain the integrity of both the communicated information and their own knowledge. Then they can successfully coordinate their activities and cooperate in solving mutual problems.

Semantic integration: Where the semantics of a resource are expressed (partially) in the form of data dictionary or schema information, this information must be interrelated with the semantics of the other resources through the use of class servers or global schemas. It is essential that a common semantics be available and provided computationally.

Database management systems for design: DBMS technology is needed that supports large and long-duration transactions, relaxed transactions, large structured composite objects, versions, and aggregation.

Intentionality: Representations for agents and their actions must be developed that can express their intentions and commitments through communicative acts.

We support the above capabilities in our architecture.

4 Semantic Integration via a Common Ontology

In order for agents to interact productively, they must have something in common, i.e., they must be either grounded in the same environment or able to relate their individual environments. We use an existing common context—the Cyc common-sense knowledge base [Lenat and Guha 1990]—to provide semantic grounding. The models of agents and resources are compared and mapped to Cyc but not to each other, making interoperation easier to attain. For *n* models, only *n* mappings are needed, instead of as many as *n(n-1)* mappings when the models are related pairwise. Currently, Cyc is the best choice for a common context, because of 1) its rich set of abstractions, which ease the process of representing predefined groupings of concepts, 2) its knowledge representation mechanisms, which are needed to construct, represent, and maintain a common context, and 3) its size: it covers a large portion of the world and the subject matter of most information resources.

The large size and broad coverage of Cyc's knowledge enable 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. Carnot can use models constructed using any of several popular formalisms, such as

- IRDS, IBM's AD/Cycle, or Bellcore's CLDM for entity-relationship models
- Ingres, Oracle, Sybase, Objectivity, or Itasca for database schemas, and
- MCC's RAD or NASA's CLIPS for agent models.

Cyc's knowledge about metamodels for these formalisms and relationships among them enables transactions to interoperate semantically between, for example, relational and object-oriented databases.

The relationship between a domain concept from a local model and one or more concepts in the common context is expressed as an articulation axiom [Guha 1990]: a statement of equivalence between components of two theories. Each axiom has the form

$$\mathrm{ist}(G; \phi) \Leftrightarrow \mathrm{ist}(C_i; \psi)$$

where ϕ and ψ are logical expressions and *ist* is a predicate that means "is true in the context." This axiom says that the meaning of ϕ in the common context G is the same as that of ψ in the local context C_i. Models are then related to each other—or translated between formalisms—via this common context by means of the articulation axioms, as illustrated in Figure 1. For example, an application's query about `Automobile` might result in subqueries to DB1 about `Car`, to DB2 about `Auto`, and to KB1 about `car`. Note that each model can be added independently, and the articulation axioms that result do not have to change when additional models are added. Also note that applications and resources need not be modified in order to interoperate in the integrated environment. We have built a graphical tool, MIST, that aids in the construction of articulation axioms.

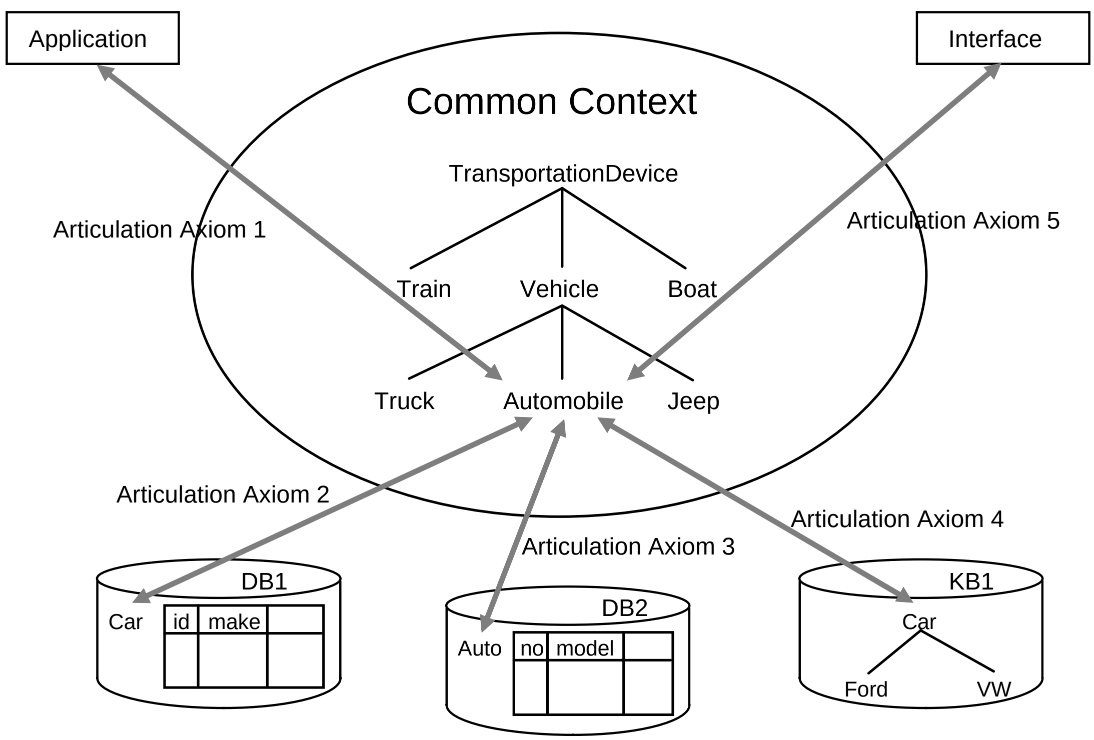

Figure 1: Concepts from different models are related via a common aggregate context by means of articulation axioms

Figure 2 shows a logical view of the execution environment. During interoperation, mediator-like agents [Wiederhold 1992], which are implemented by Rosette actors

[Tomlinson et al. 1991], apply the articulation axioms that relate each agent or resource model to the common context. This performs a translation of message semantics. At most *n* sets of articulation axioms and *n* agents are needed for interoperation among *n* resources and applications. The agents also apply a syntax translation between each local data-manipulation language, DML_i, and the global context language, *GCL*. *GCL* is based on extended first-order logic. A local data-manipulation language might be, for example, SQL for relational databases or OSQL for object-oriented databases. The number of language translators between DML_i and *GCL* is no greater than *n*, and may be a constant because there are only a small number of data-manipulation languages that are in use today. Additional details describing how transactions are processed semantically through the global and local views of several databases can be found in [Woelk *et al.* 1996].

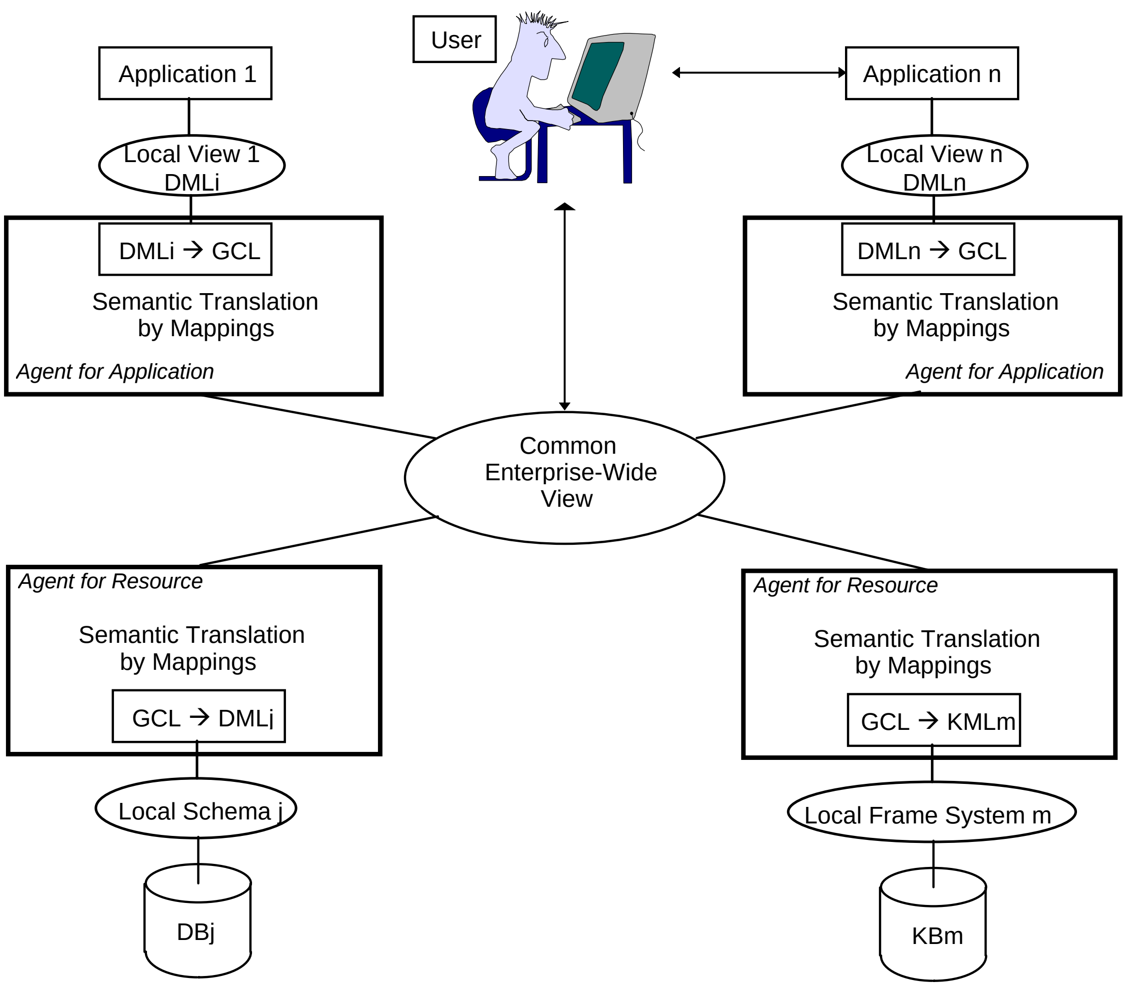

Figure 2: Logical view of the execution environment, showing how mediating agents apply articulation axioms to achieve semantic interoperation

The agents also function as communication aides, by managing communications among the databases, application programs, and other agents in the environment. Figure 3 shows how they buffer messages, locate message recipients, and translate message semantics. To implement message transfer, they use a tree-space mechanism—a kind of distributed virtual blackboard—built on the OSI and TCP/IP protocols [Tomlinson *et al*. 1991].

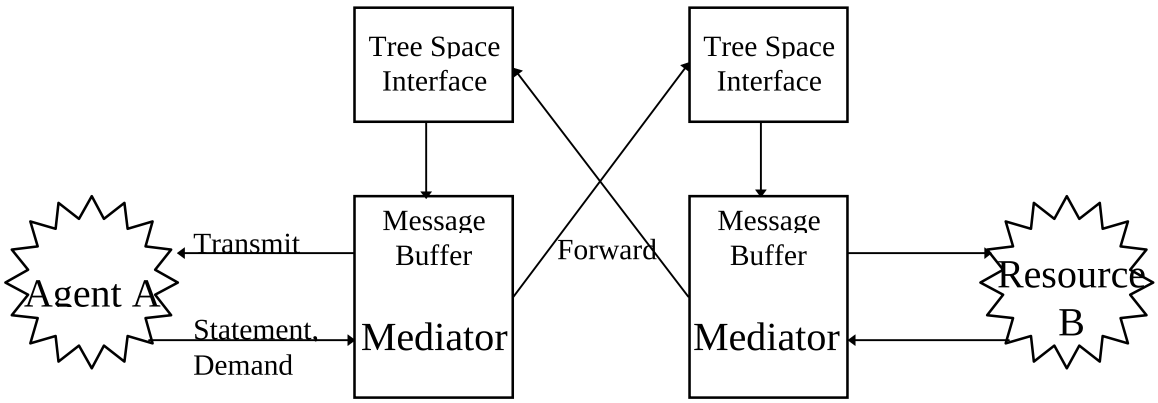

Figure 3: Each database, application, and reasoning agent has a mediator (actor) that manages its communications through a tree space.

5 Application to Parameterized System Design

We have applied our methodology to automate parameterized design of telecommunication services, the task of providing communication facilities to customers and often called provisioning. This task is executed in a heterogeneous multidatabase environment. It is an example of workflow control, in that it provides control and data flows among transactions executing on multiple autonomous systems [Jin *et al*. 1993; Tomlinson *et al*. 1993]. Service provisioning typically takes several weeks and requires coordination among many operation-support systems and network elements. Configuring the operation-support systems so that they can perform such a task often takes several months to complete.

We investigated ways to improve the provisioning of one type of communication facility—digital services (DS-1). Provisioning DS-1 takes more than two weeks and involves 48 separate operations—23 of which are manual—against 16 different database systems. Our goals were to reduce this time to less than two hours and to provide a way in which new services could be introduced more easily. Our strategy for accomplishing these goals was to 1) interconnect and interoperate among the previously independent systems, 2) replace serial operations by concurrent ones by making appropriate use of relaxed transaction processing [Attie *et al*. 1993; Bukhres *et al*. 1993; Elmagarmid 1992; Ansari *et al*. 1992], and 3) automate previously manual operations, thereby reducing the incidence of errors and delays. The transaction processing is relaxed in that some subsystems are allowed to be temporarily inconsistent, although eventual consistency is guaranteed. Relaxing the consistency requirements allows increased concurrency and, thus, improved throughput and response time.

The architecture of the agents used to implement relaxed transaction processing is shown in Figure 4. The agents operate as follows. The graphical-interaction agent helps a user fill in an order form correctly, and checks inventories to give the user an estimate of when the order will be completed. It also informs the user about the progress of the order.

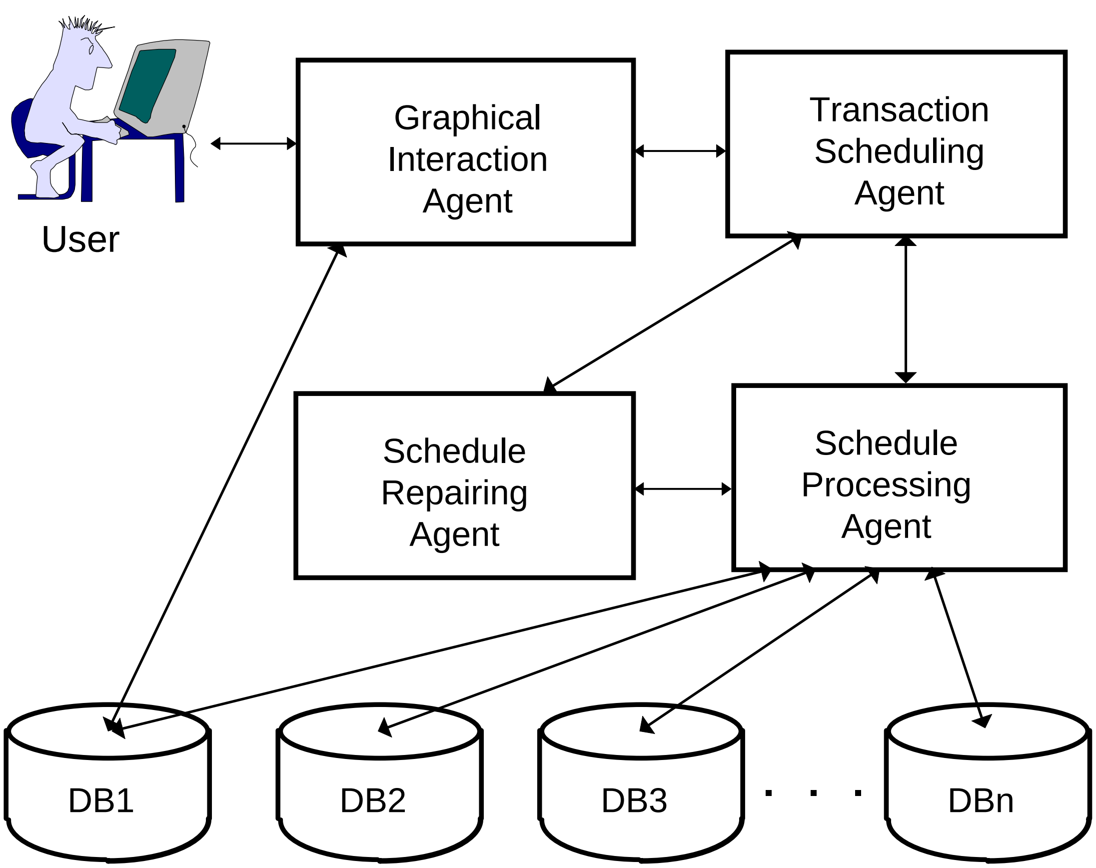

Figure 4: Multiagent system for relaxed processing of telecommunication transactions

The transaction-scheduling agent constructs the schedule of tasks needed to satisfy an order. The tasks are scheduled with the maximum concurrency possible, while still satisfying their precedence constraints. Some of the rules that implement the schedule are shown in Figure 5. These particular rules, when appropriately enabled, generate a subtransaction to update the database for customer billing. When executing such rules, the transaction-scheduling agent behaves as a finite-state automaton, as shown in Figure 6. The resultant schedule showing the commit dependencies among the tasks for all such automata is shown in Figure 7.

```
; This rule set 1) executes an external program that translates an Access
; Service Request into a command file to update the database for customer
; billing, 2) executes the command file, and 3) checks for completion.
; The scheduling agent, due to its truth-maintenance system, halts this
; transaction whenever an abort of the global transaction occurs.
; ?gtid denotes the global transaction identifier.
Bill-Preparation:
  If   (service-order(?gtid)
        new-tid(?subtid)
        unless(abort(?gtid)))
  then (do(,run-shell-program "asr2bill"
              :input "asr-?gtid.out"  :output "bill-?gtid.sql")
        bill(?gtid ?subtid)
        tell(GIAgent "task ?gtid BILLING ready"))

Bill-Execution:
  If   (bill(?gtid ?subtid)
        logical-db(?db))
  then (tell(SchedProcAgent
               "task-execute ?subtid BILL ?db bill-?gtid.sql")
        tell(GIAgent "task ?gtid BILLING active"))

Bill-Completion:
  If   (success(?subtid)
        bill(?gtid ?subtid))
  then (tell(GIAgent "task ?gtid BILLING done"))

Bill-Failure:
  If   (failure(?subtid)
        excuse(bill(?gtid ?subtid)))
  then (abort(?gtid)
        tell(GIAgent "task ?gtid BILLING failed"))
```

Figure 5: Rules used by the transaction-scheduling agent to generate a workflow schedule

The schedule-processing agent maintains connections to the databases involved in telecommunication provisioning, and implements transactions on them. It knows how to construct the proper form for a transaction, based on the results of other transactions. The transactions are processed concurrently, where appropriate. If something goes wrong during the processing of a transaction that causes it to abort or fail to commit, the schedule-repairing agent provides advice on how to fix the problem and restore consistency. The advice can be information on how to restart a transaction, how to abort a transaction, how to compensate for a previously committed transaction, or how to clean-up a failed transaction. The integrity knowledge that is stored in the schedule repairing agent comes from a comparison of the models, as expressed in terms of the common ontology.

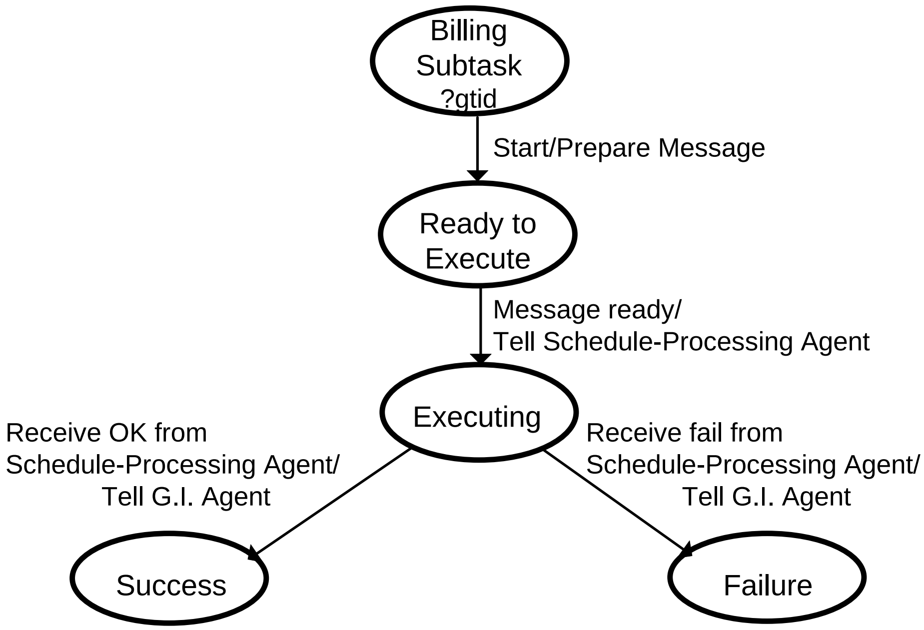

Figure 6: Representative finite-state automaton for a telecommunication service provisioning task assigned by the transaction-scheduling agent

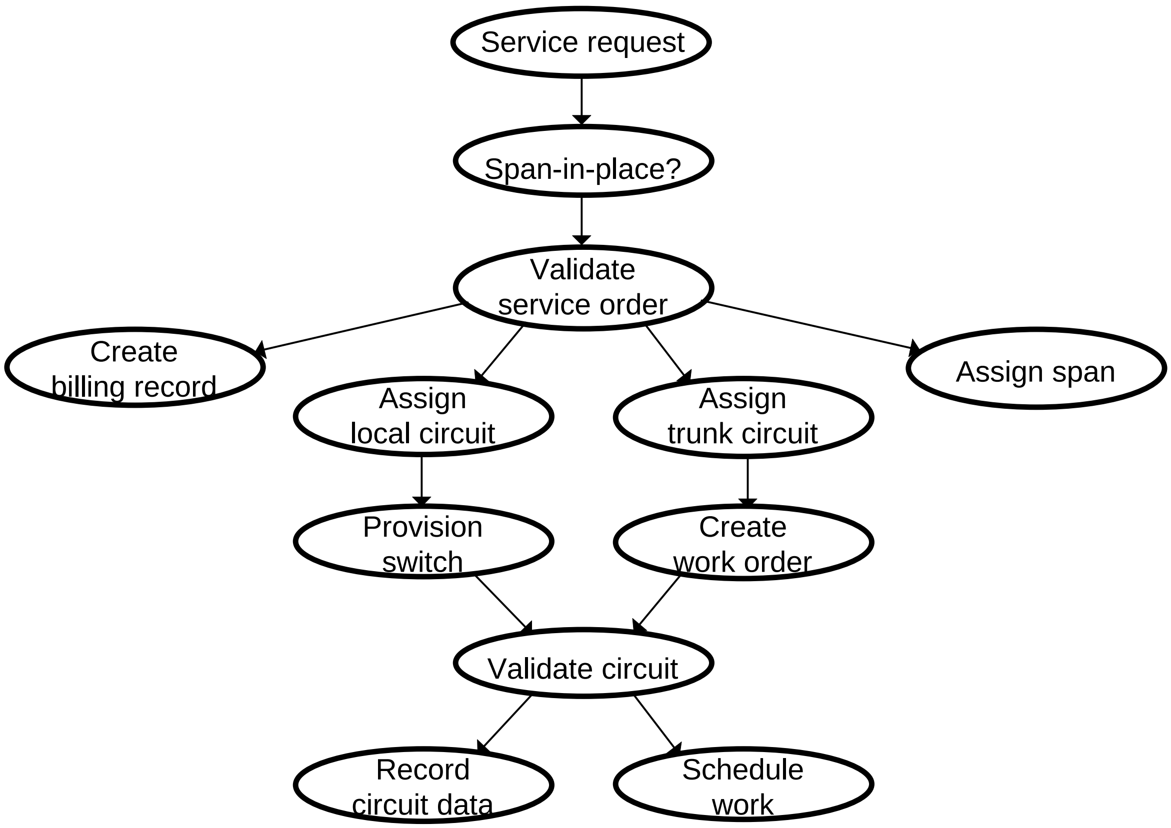

Figure 7: Workflow for telecommunication service provisioning generated by the transaction-scheduling agent. Only the default workflow is shown, without any exception paths.

The agents, as described above, are simply expert systems whose expertise is in processing orders for telecommunication services. However, they have the additional abilities to interact and cooperate with each other. Their interaction is via the mediators shown in Figure 3.

The agents cooperate, at the knowledge level [Newell 1982], via models of themselves. For example, a conceptual domain model for the graphical-interaction agent is shown in Figure 8. An interface form that provides user access and modifications to the knowledge possessed by this agent is shown in Figure 9. Entries on the form, or the form's completion, cause queries and transactions to be sent to the other agents or databases in the environment. Note, however, that the model does not capture the procedural knowledge necessary to specify the queries and transactions; a technique for modeling processes is needed to capture such knowledge. In other words, the models represent the static knowledge of the agents, not (unfortunately) their dynamics. Nevertheless, they have proven useful in enabling the agents to interact coherently, as we describe next.

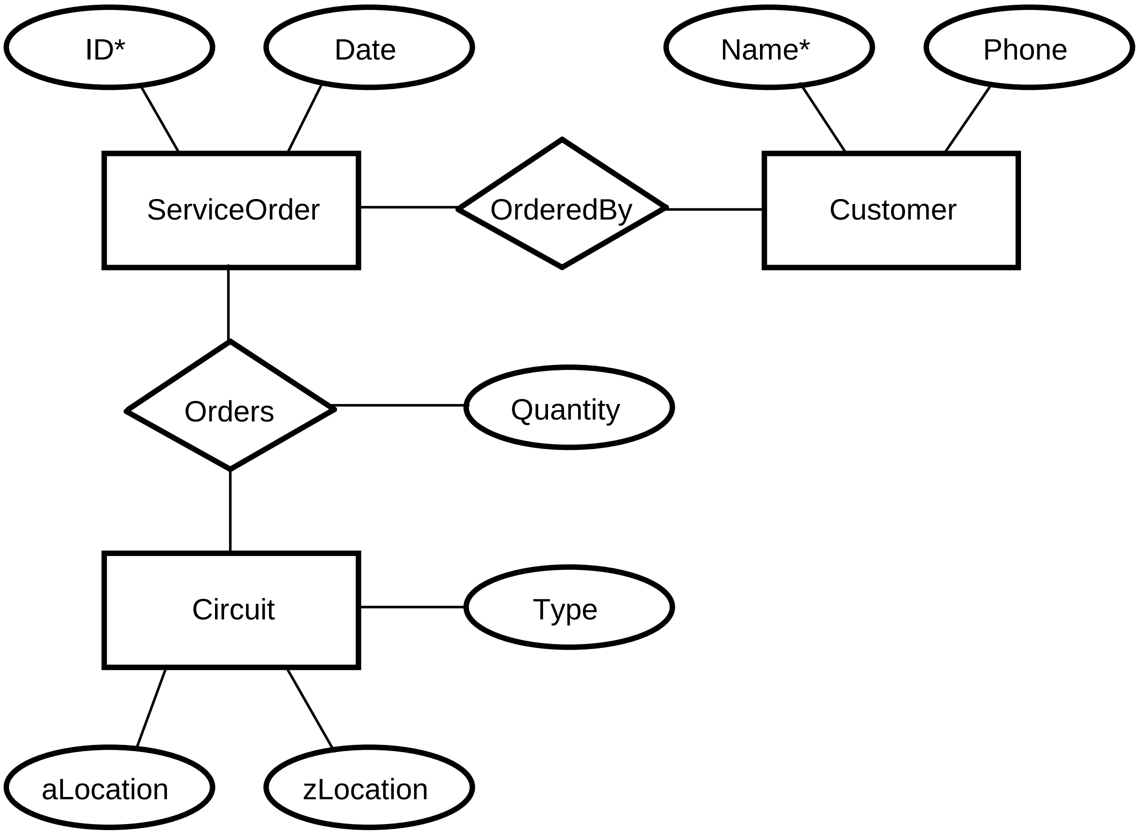

Figure 8: Semantic model (simplified) for the graphical-interaction agent

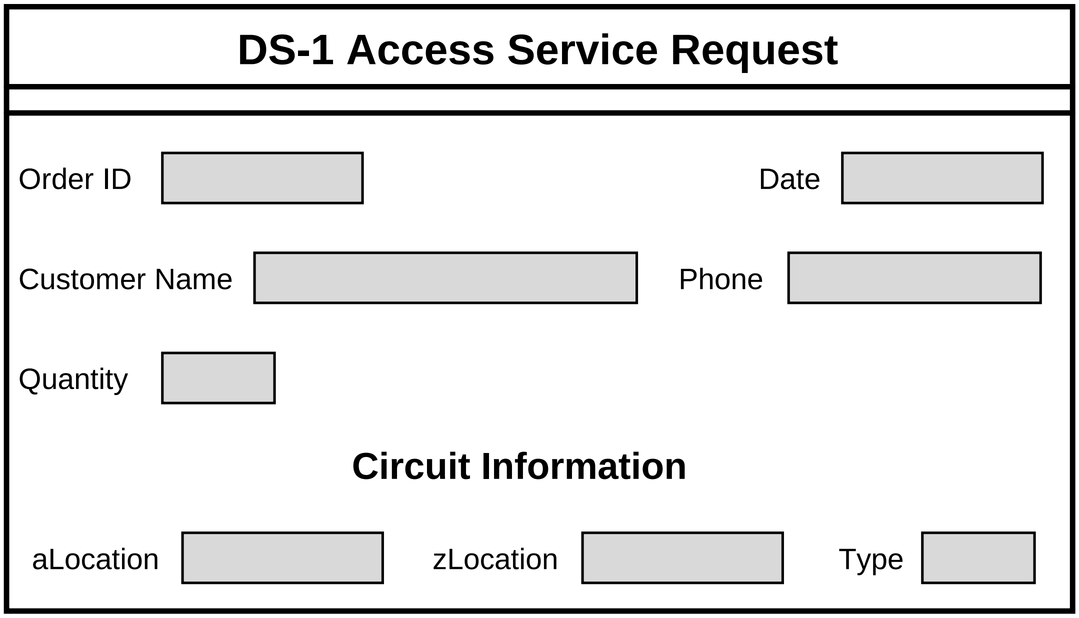

Figure 9: User interface form (simplified) corresponding to the declarative knowledge of the graphical-interaction agent

Conceptual models for two more of the agents are shown in Figures 10 and 11. Each model consists of organized concepts describing the context, domain, or viewpoint of the knowledge possessed by that agent, i.e., the knowledge base of each agent contains rules written in terms of these concepts.

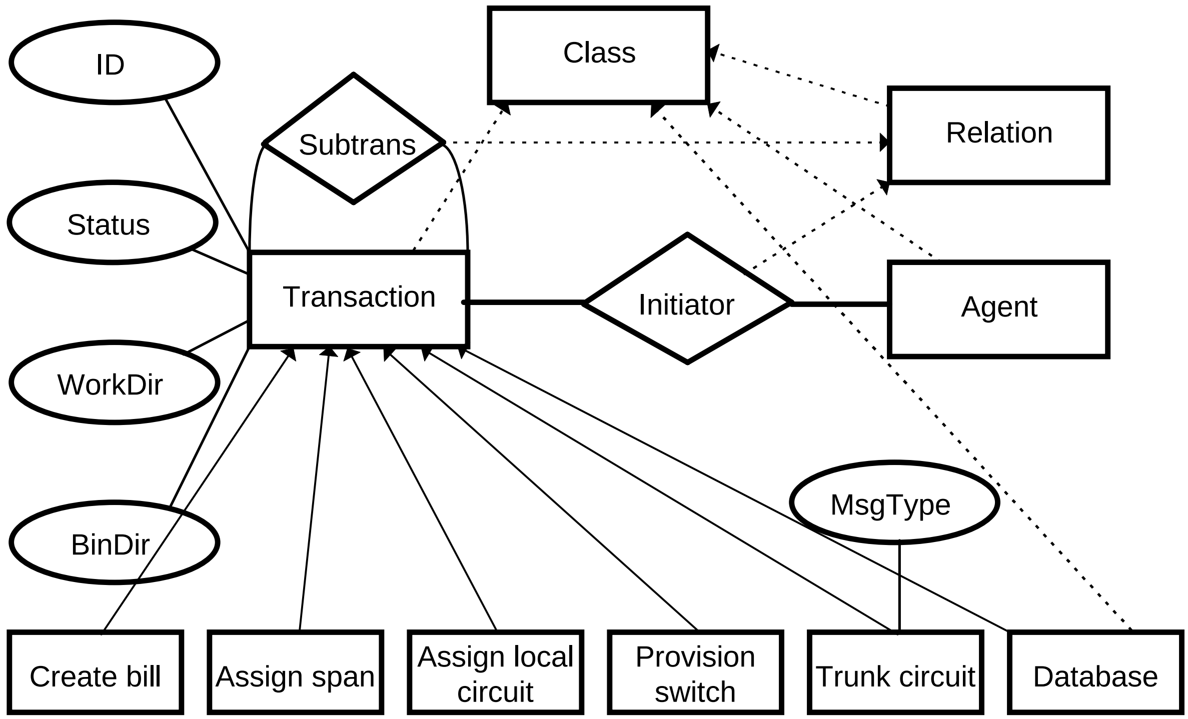

Figure 10: Semantic model for the transaction-scheduling agent (dashed arrows indicate instance relationships, and solid arrows indicate subclass relationships)

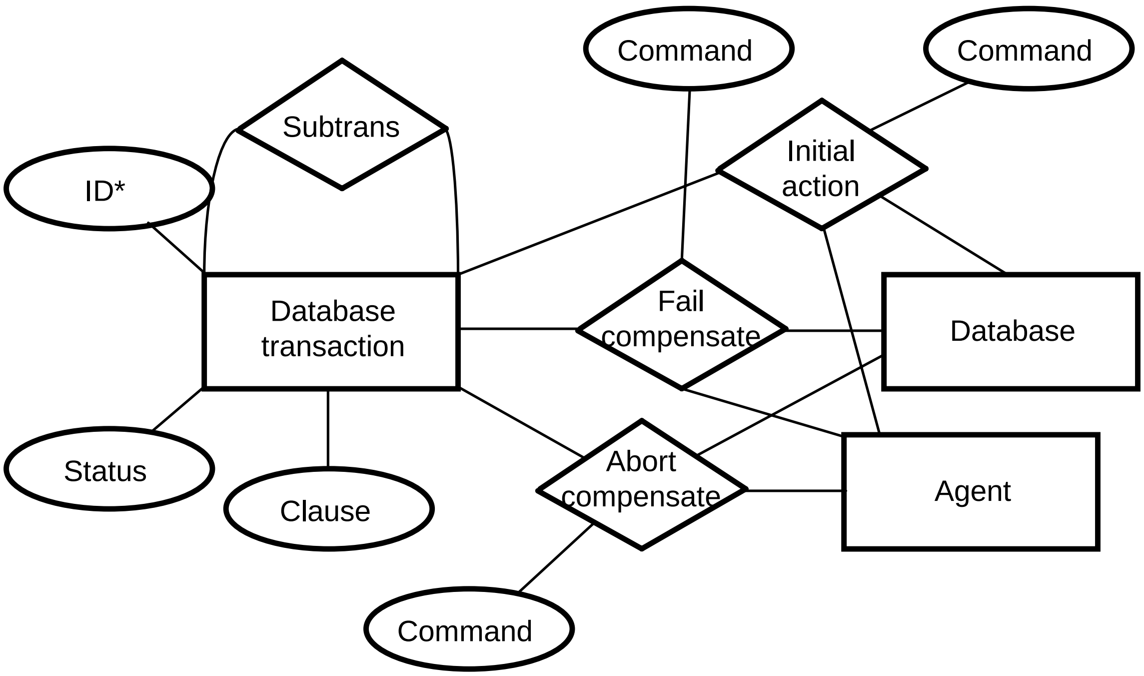

Figure 11: Semantic model for the schedule-repairing agent

All of the models in Figures 8, 10, and 11 are related to the common context, and thereby to each other, via articulation axioms. For example, the concept `Transaction` for the transaction-scheduling agent and the concept `DBTransaction` for the schedule-repairing agent are each related to the common concept `DatabaseTransaction` via the axioms

$$ist(Cyc, DatabaseTransaction(T)) \Leftrightarrow ist(Scheduler, Transaction(T))$$

$$ist(Cyc, DatabaseTransaction(T)) \Leftrightarrow ist(Repairer, DBTransaction(T))$$

The axioms are used to translate messages exchanged by the agents, so that the agents can understand each other. In the above example, the two agents would use their axioms to converse about the status of database transactions, without having to change their internal terminology. Similar axioms describing the semantics of each of the databases involved enable the schedule-processing agent to issue transactions to the databases. The axioms also relate the semantics of the form shown in Figure 9 to the semantics of the other information resources in the environment. Such axioms are constructed with the aid of a graphical tool called MIST, for Model Integration Software Tool. The operation of MIST is described in the Appendix.

Operationally, the axioms are managed and applied by the mediators that assist each agent. They use the axioms to translate each outgoing message from their agent into the common context, and to translate each incoming message for their agent into its local semantics.

6 Background and Discussion

Integrating enterprise models is similar to integrating heterogeneous databases. Two approaches have been suggested previously for this [Buneman *et al*. 1990]. The *composite approach* produces a global schema by merging the schemas of the individual databases. Explicit resolutions are specified in advance for any semantic conflicts among the databases, so users and applications are presented with the illusion of a single, centralized database. However, the centralized view may differ from the previous local views and existing applications might not execute correctly any more. Further, a new global schema must be constructed every time a local schema changes or is added.

The *federated approach* [Heimbigner and McLeod 1985, Litwin *et al*. 1990] 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 [Ahlsen and Johannesson 1990]. 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 common schema.

We base our methodology on the composite approach, but make three changes that enable us to combine the advantages of both approaches while avoiding some of their shortcomings. First, we use an *existing* common schema or context. In a similar attempt, [Sull and Kashyap 1992] describes a method for integrating schemas by translating them into an object-oriented data model, but this method maintains only the structural semantics of the resources.

Second, we capture the mapping between each model and the common context in a set of articulation axioms. 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.

Third, we consider knowledge-based systems (KBSs), interfaces, and applications, as well as databases.

Our use of agents for interoperating among applications and information resources is similar to the uses of mediators described in [Wiederhold 1992]. However, we also specify a means for semantic translation among the agents, as well as an implemented prototype. Other applications of similar agents, such as the Pilot's Associate developed by Lockheed *et al*. [Smith and Broadwell 1988], handcrafted their agents. This is not possible for large “open” applications: the agents must be such that they can be developed independently and execute autonomously.

Our architecture employs two kinds of computational agents: finer-grained, concurrent actors and coarser-grained, knowledge-based systems. The actors are used to control interactions among the components of the architecture. The knowledge-based agents are used where reasoning is needed, such as in deciding what tasks should be performed next or how to repair the environment when a task has failed. This seems to be a natural division of responsibilities for our example application. However, we took an engineering, rather than a scientific, approach, in that we did not investigate any alternative architectures.

7 Conclusion

For years, information-system personnel managed corporate data that was centralized on mainframes. The data was kept consistent, but eventually the amount of data increased to the point that centralized storage was no longer viable. Also, users wanted a way to share data across applications and wanted more direct involvement in the management of the data. So, data then began proliferating onto workstations and personal computers, where users could manage it themselves. But this resulted in redundancy, inconsistency, and no coherent global view. Hence, there are now attempts to reintegrate data. Users still need to manage their own data, which remains distributed, but they and their applications need coherent global access and consistency must be restored.

This paper describes Carnot's approach to enabling interoperation among enterprise information objects, i.e., among suppliers and consumers of information. In this approach, an enterprise information object is integrated based on articulation axioms defined between two contexts: the context of a model of the object and a common context provided by the Cyc knowledge base. The methodology is based on the following principles:

- Existing information resources should not have to be modified and data should not have to migrate.

- Existing applications should not have to be modified.

- Users should not have to adopt a new language for communicating with the resultant integrated system, unless they are accessing new types of information.

- Resources and applications should be able to be integrated independently, and the mappings that result should not have to change when additional objects are integrated.

The above principles are incorporated in an integration tool, MIST, for assisting an administrator in generating articulation axioms for a model, and in a set of agents that utilize the resultant axioms to provide users and applications with access to the integrated resources. They can use a familiar local context, while still benefiting from newly added resources. These systems constitute part of the semantic services of Carnot [Cannata 1991]. They help specify and maintain the semantics of an organization's integrated information resources.

Extensions of our work are focused on developing additional information-system applications for agents, including intelligent directory service agents, negotiating electronic data interchange (EDI) agents, database administration agents, and intelligent information retrieval agents. Our most important future work is centered on ways in which agents can acquire and maintain models of each other in order to improve their interactions.

References

[Agha 1986] Gul Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, MA, 1986.

[Ahlsen and Johannesson 1990] 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*. 1992] Mansoor Ansari, Marek Rusinkiewicz, Linda Ness, and Amit Sheth, "Executing Multidatabase Transactions," *Proceedings 25th Hawaii International Conference on Systems Sciences*, January 1992.

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

[Bukhres *et al*. 1993] 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*. 1990] O. P. Buneman, S. B. Davidson, and A. Watters, “Querying Independent Databases,” *Information Sciences*, Vol. 52, Dec. 1990, pp. 1–34.

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

[Ceri and Widom 1992] Stefano Ceri and Jennifer Widom, “Production Rules in Parallel and Distributed Database Environments,” *Proceedings of the 18th VLDB Conference*, Vancouver, 1992, pp. 339–351.

[Collet *et al*. 1991] Christine Collet, Michael N. Huhns, and Wei-Min Shen, “Resource integration using a large knowledge base in Carnot,” *IEEE Computer*, Vol. 24, No. 12, Dec. 1991, pp. 55–62.

[Elmagarmid 1992] Ahmed Elmagarmid, ed., *Database Transaction Models*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Gasser and Huhns 1989] Les Gasser and Michael N. Huhns, eds., *Distributed Artificial Intelligence, Volume II*, Pitman Publishing, London, 1989.

[Guha 1990] R. V. Guha, “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.

[Heimbigner and McLeod 1985] 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 1987] Michael N. Huhns, ed., *Distributed Artificial Intelligence*, Pitman Publishing, London, 1987.

[Huhns and Bridgeland 1991] Michael N. Huhns and David M. Bridgeland, “Multiagent Truth Maintenance,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 6, December 1991, pp. 1437–1445.

[Huhns and Singh 1997] Michael N. Huhns and Munindar P. Singh, eds., *Readings in Agents*, Morgan Kaufmann Publishers, San Francisco, 1997.

[Jin *et al*. 1993] W. Woody Jin, Linda Ness, Marek Rusinkiewicz, and Amit Sheth, "Executing Service Provisioning Applications as Multidatabase Flexible Transactions," Bellcore Technical Report (unpublished), 1993.

[Lenat and Guha 1990] Doug Lenat and R. V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*, Addison-Wesley, Reading, MA, 1990.

[Litwin *et al.* 1990] Witold Litwin, Leo Mark, and Nick Roussopoulos, "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, Sept. 1990, pp. 267–296.

[Newell 1982] Allen Newell, "The Knowledge Level," *Artificial Intelligence*, Vol. 18, No. 1, January 1982, pp. 87–127.

[Sheth and Larson 1990] Amit P. Sheth and James A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, Sept. 1990, pp. 183–236.

[Singh *et al*. 1997] Munindar P. Singh, Philip E. Cannata, Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Kayliang Ong, Amit Sheth, Christine Tomlinson, and Darrell Woelk "The Carnot Heterogeneous Database Project: Implemented Applications," *Distributed and Parallel Databases: An International Journal*, Vol. 5, No. 2, April 1997, pp. 207-225.

[Smith and Broadwell 1988] David Smith and Martin Broadwell, "The Pilot's Associate—an overview," *Proceedings of the SAE Aerotech Conference*, Los Angeles, CA, May 1988.

[Sull and Kashyap 1992] Wonhee Sull and Rangaswami L. Kashyap, "A Self-Organizing Knowledge Representation Scheme for Extensible Heterogeneous Information Environment," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 2, April 1992, pp. 185–191.

[Tomlinson *et al*. 1991] Chris Tomlinson, Mark Scheevel, and Vineet Singh, "Report on Rosette 1.1," MCC Technical Report Number ACT-OODS-275-91, Microelectronics and Computer Technology Corporation, Austin, TX, July 1991.

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

[Wiederhold 1992] Gio Wiederhold, “Mediators in the Architecture of Future Information Systems,” *IEEE Computer*, Vol. 25, No. 3, March 1992, pp. 38–49.

[Woelk *et al*. 1996] Darrell Woelk, Philip E. Cannata, Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Greg Lavender, Greg Meredith, Kayliang Ong, Wei-Min Shen, Munindar P. Singh, and Christine Tomlinson, “Carnot Prototype,” in Omran A. Bukhres and Ahmed K. Elmagarmid, eds., *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*, Prentice-Hall, 1996, pp. 621-648.

Appendix: The Development of Articulation Axioms

Carnot provides a graphical tool, the Model Integration Software Tool (MIST), that 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:

1. MIST automatically represents an enterprise model in a local context as an instance of a given formalism. The representation is declarative, and uses an extensive set of semantic properties.

2. By constraint propagation and user interaction it matches concepts from the local context with concepts from the common context.

3. For each match, it automatically constructs an articulation axiom by instantiating axiom templates.

MIST displays enterprise models both before and after they are represented in a local context. MIST enables a global knowledge base, representing a common enterprise-wide context, to be browsed graphically and textually to allow the correct concept matches to be located. With MIST, a user to create frames in the common 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 that it constructs. Figure 12 illustrates the components of an information environment that MIST uses in producing articulation axioms. The three phases of articulation axiom development are described next in more detail.

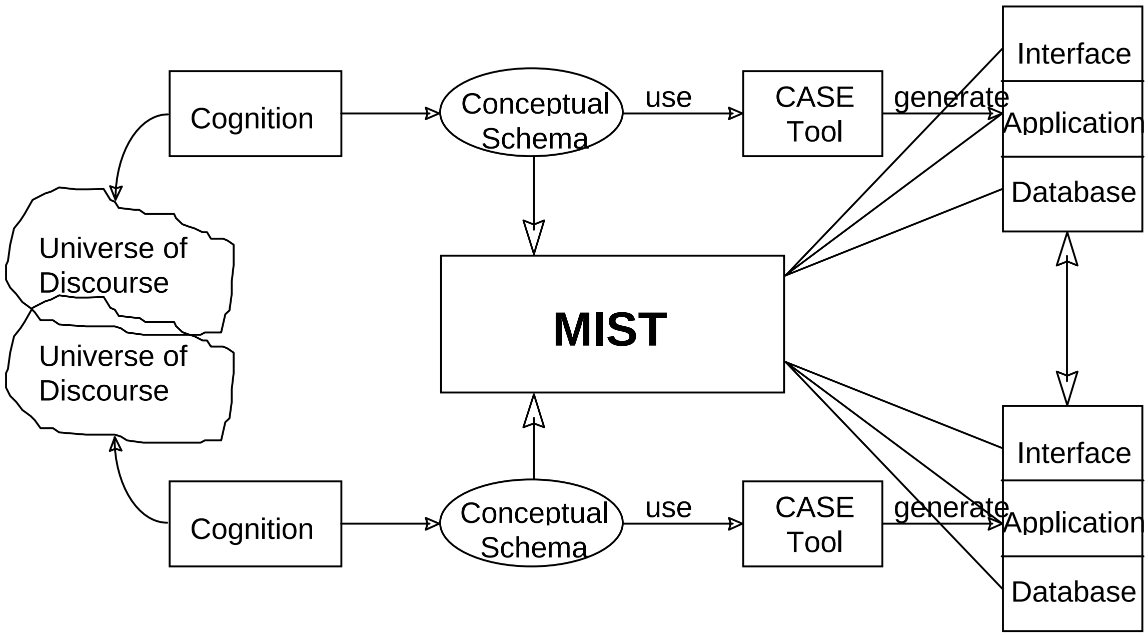

Figure 12: MIST relates the conceptual schemas of information systems, thereby aiding in the development of articulation axioms that enable the information systems to interoperate

In the model representation phase, we represent the model as a set of frames and slots in a context created specially for it. These frames are instances of frames describing the metamodel of the schema, e.g., `Relation` and `DatabaseAttribute` for a relational schema.

In the matching phase, the problem is: given a representation for a concept in a local context, find its corresponding concept in the common context. The two factors that affect this phase are 1) there may be a mismatch between the local and common contexts in the depth of knowledge representing a concept, and 2) there may be mismatches between the structures used to encode the knowledge. For example, a concept in Cyc can be represented as either a collection or an attribute [Lenat and Guha 1990, pp. 339ff].

If the common 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 common context's knowledge. If the common context has less knowledge than the local context, then knowledge will be added to the common context until its knowledge equals or exceeds that in the local context; otherwise, the common context would be unable to model the semantics of the resource. The added knowledge refines the common 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 common 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 common context. Matching begins by finding associations between attribute/link definitions and existing slots in the common 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.

In the third phase, 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 common schema, and vice versa. Articulation axioms are generated automatically by instantiating stored templates with the matches found.