MCC Technical Report Number Carnot-148-93

Global Information Management via Local Autonomous Agents

Michael N. Huhns, Munindar P. Singh,
Tomasz Ksiezyk, and Nigel Jacobs

November 1993

MCC/ISD Nonconfidential

Abstract

In this report 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, domain models of business environments, or process models of business operations. Models enable the agents and information resources to use the appropriate semantics when they communicate with each other. This is accomplished by specifying the semantics in terms of a common, aggregate 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. The agents implement virtual state machines, and interact by exchanging state information. The resultant interaction produces an implementation of relaxed transaction processing.

Microelectronics and Computer Technology Corporation
Information Systems Division
3500 West Balcones Center Drive
Austin, TX, U.S.A. 78759-6509
(512) 338-3651 or huhns@mcc.com

1 Introduction

Business operations, including sales, marketing, manufacturing, and design, 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.

Some of the techniques for dealing with the size and complexity of these enterprise information systems are modularity, distribution, abstraction, and intelligence, i.e., being smarter about how you seek and modify information. Combining these techniques implies the use of intelligent, distributed modules—a distributed artificial intelligence approach. In accord with this approach, we 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 report.

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 in an enterprise and how agents can acquire them. We use the ontology developed for the large knowledge-based system, Cyc, for semantic grounding of the models. This provides a common ontology. We then describe a set of agents for telecommunication service provisioning—a scheduling agent, a schedule-repairing agent, a schedule-processing agent, and an interface agent—and describe their models and how they use them to cooperate. We also describe the use of actors [Agha 1986]—one per agent—who manage communications among the agents. Each actor independently maintains the relationship between its agent and the common ontology (in the form of articulation axioms), and updates that relationship as the ontology changes or the agent itself evolves.

- 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 and there are instead 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 underlying hardware and operating systems, in data structures, and in 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.

- MCC's RAD or NASA's CLIPS for agent models.

Cyc's knowledge about metamodels for these formalisms and the 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 $ist(G\ \phi) \Leftrightarrow ist(C_i\ \psi)$, 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` would 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. The Appendix contains a description of the graphical tool, MIST, that we have built to aid in the construction of articulation axioms.

Figure 2 shows a logical view of the execution environment. During interoperation, mediators [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 mediators are needed for interoperation among n resources and applications. The mediators also apply a syntax translation between a 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.* 1992].

The mediators also function as communication aides, by managing commu-

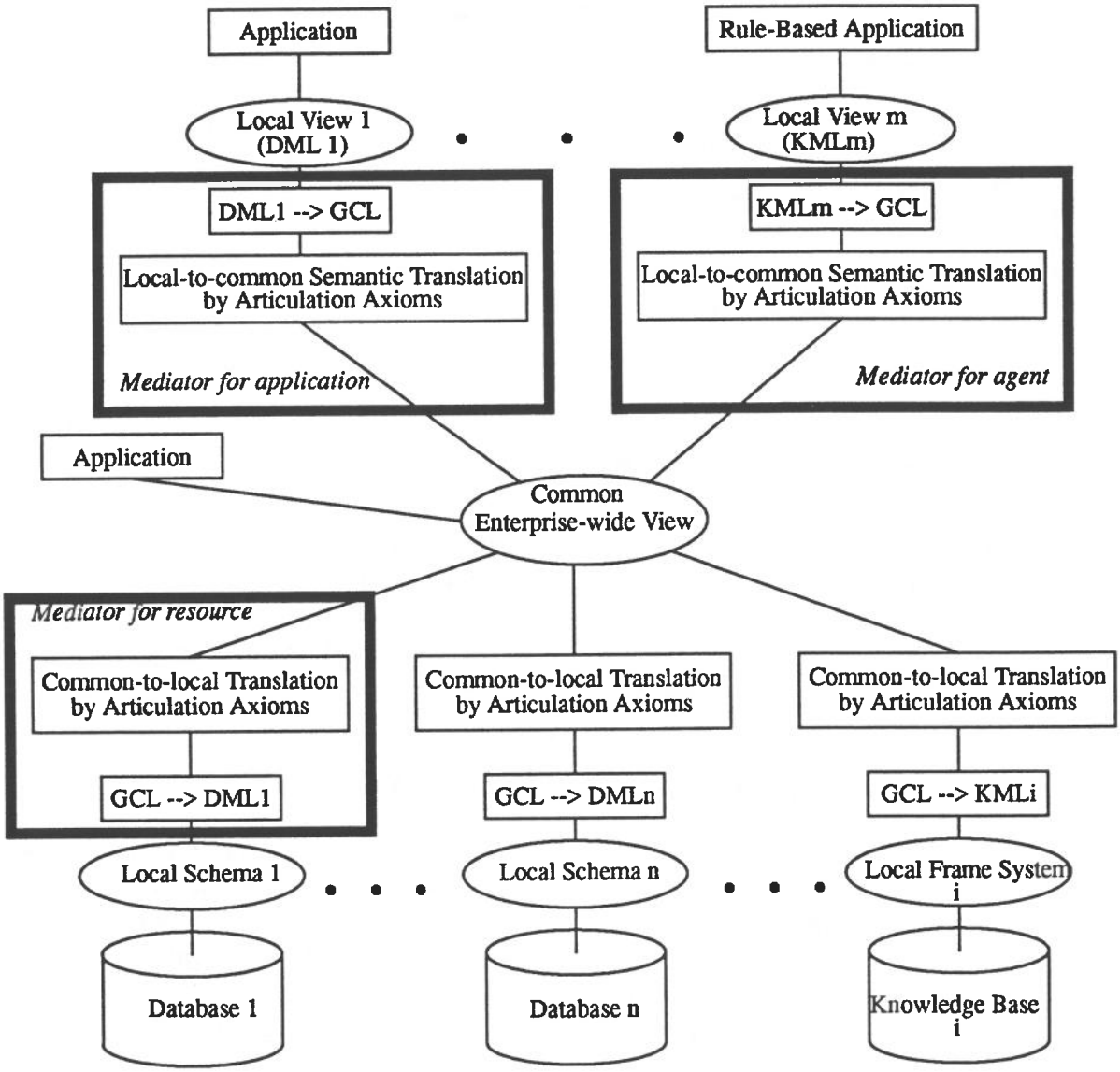

Figure 2: Logical view of the execution environment, showing how mediating agents apply articulation axioms to achieve semantic interoperation

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 [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.

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 precedence constraints among themselves. 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.

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.

```
;;; Execute an external program that translates an Access Service Request
;;; into a command file to update the database for customer billing.  Execute
;;; the command file and then check for completion.  Note that the scheduling
;;; agent, due to its truth-maintenance system, stops processing this
;;; subtransaction whenever an abort of the global transaction occurs.
;;; ?gtid denotes the global transaction identifier.
BILL-prepare:
  If    (service-order(?gtid)
         new-tid(?subtid)
         unless(abort(?gtid)))
  then (do(,run-shell-program
              ("asr2bill"
               :input ("asr-?gtid.out")
               :output "cabs-?gtid.sql"))
         bill(?gtid ?subtid)
         tell(GIAgent "Task ?gtid BILLING ready"))

BILL-execute:
  If    (bill(?gtid ?subtid)
         logical-db(?db))
  then (tell(SchedProcAgent "task-execute ?subtid BILL ?db cabs-?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: Some of the rules used by the transaction-scheduling agent to generate a schedule for DS-1 workflow

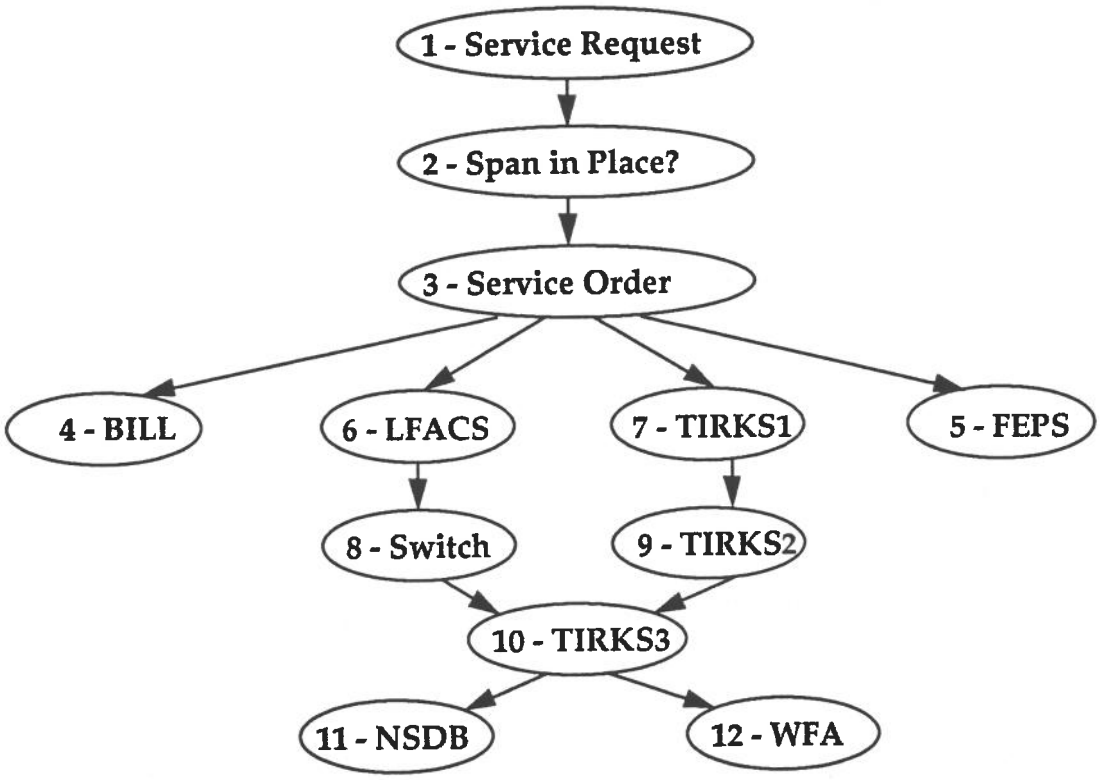

Figure 7: Workflow for telecommunication service provisioning generated by the transaction-scheduling agent. Note that multiple such service requests can be processed concurrently

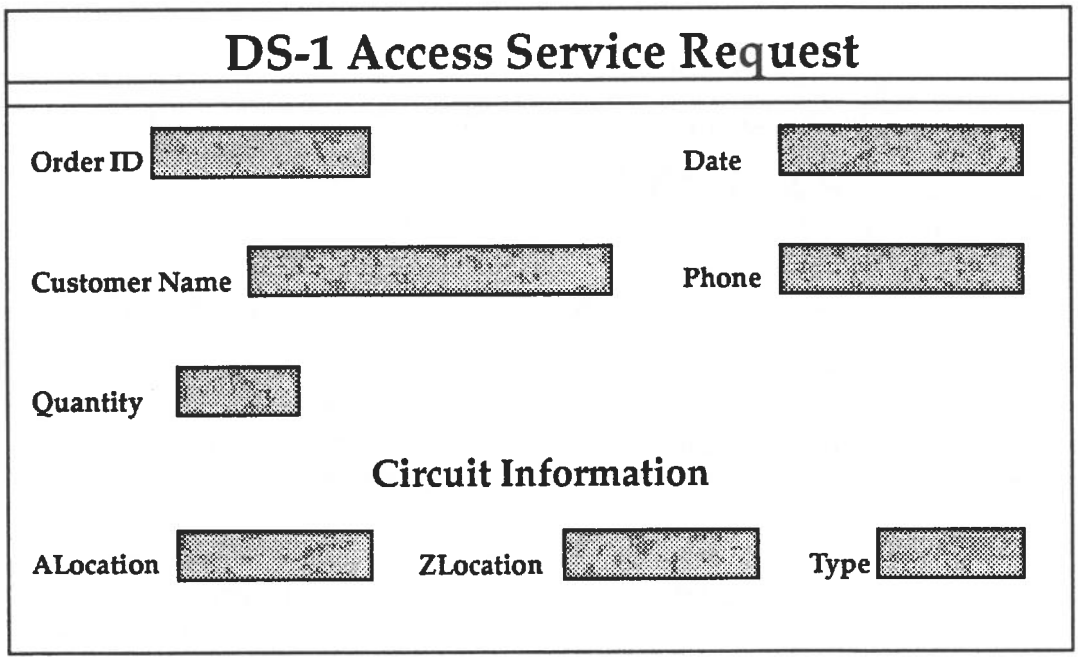

Figure 9: User interface form (simplified) corresponding to the declarative knowledge of the graphical-interaction agent

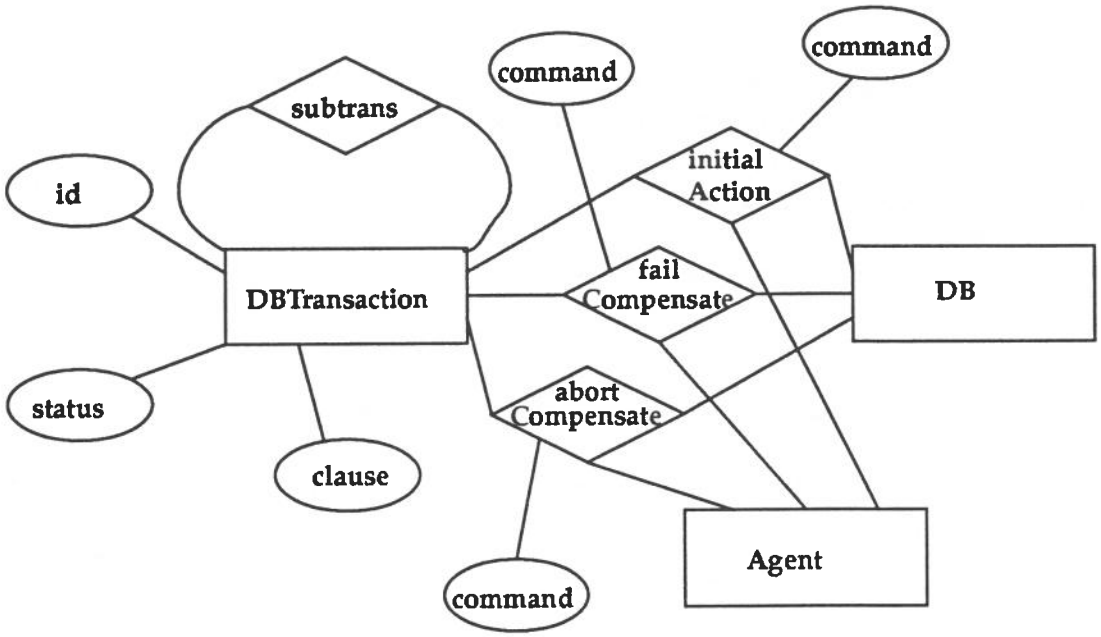

Figure 11: Semantic model for the schedule-repairing agent

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), process models, 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.

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 triggers—making passive databases active
- rule-based database applications
- database administration agents
- 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*, Janaury 1992.

[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.

[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.

[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 Rangasami 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.

[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.* 1992] Darrell Woelk, Wei-Min Shen, Michael N. Huhns, and Philip E. 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.

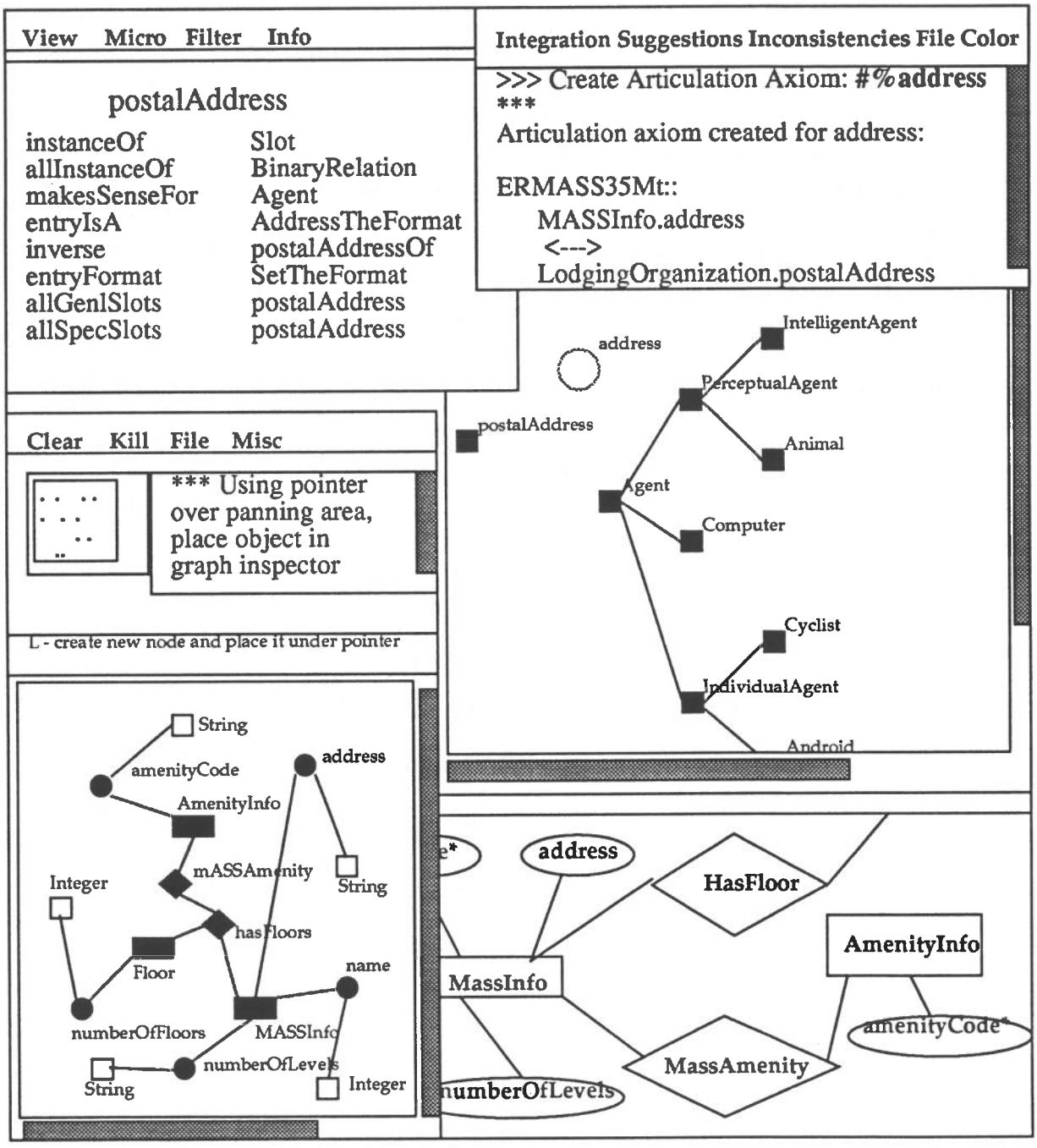

Figure 12: MIST displays an information model both before and after it is integrated, and shows the mappings (articulation axioms) that it constructs