Multiagent Systems in Information-Rich Environments

Michael N. Huhns[1]* and Munindar P. Singh[2]**

[1] Department of Electrical and Computer Engineering
University of South Carolina
Columbia, SC 29208, USA
huhns@sc.edu
[2] Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534, USA
singh@ncsu.edu

Abstract. Information-rich environments are the open environments that characterize most of the modern applications of computing technology. The applications include ubiquitous information access, electronic commerce, virtual enterprises, logistics, and sensor integration, to name but a few. These applications differ from conventional database applications not only in the nature and variety of information they involve, but also in including a significant component that is beyond the information system *per se*: the creation, transformation, use, and ultimate fate of information. The environments are typified only by the large amounts and varieties of information they include, and whose effective and efficient management is key to the above applications. Multiagent systems (MAS) are an important paradigm for building complex information systems, especially cooperative ones. We describe how cooperative information system architectures have evolved a set of common types of computational agents. We also describe two approaches that address complementary aspects of MAS construction. These approaches may be considered as components of our ongoing research program on *interaction-oriented programming*.

1 Introduction

Agents and multiagent systems (MAS) are an emerging paradigm for software development, especially of the next-generation large-scale information systems known as cooperative information systems (CIS) [6]. CIS is a research area that syntheszes results from databases and artificial intelligence. It is the study of multiagent systems and organizational and database abstractions geared toward the large, open, heterogeneous information environments of today. CIS occur

* Supported by the Defense Advanced Research Projects Agency.
** Supported by the NCSU College of Engineering, the National Science Foundation under grants IRI-9529179 and IRI-9624425 (Career Award), and IBM corporation.

in several important applications, such as enterprise integration and electronic commerce.

Numerous definitions of agents are known in the literature [3, 5, 8]. Indeed, the only agreement seems to be that there is a range of definitions! Some of the important properties of agents include autonomy, adaptability, and interactiveness, but exceptions reveal that an agent need not have these properties. However, we believe that agents *must* be capable of interacting with other agents at the social or communicative level. We distinguish social or communicative interactions from incidental interactions that agents may have as a consequence of existing in a shared environment.

Indeed, we would go so far as to claim that the potential for participation in a multiagent system is an essential property of agents. We defend this view more carefully in [4, 5]. In general, we take the view, in sympathy with Wegner [12], that interaction is a key extension beyond traditional computer science. In broad terms, the goal of this paper is to outline what interaction primitives are suitable for multiagent systems, and how they may be incorporated in designing and implementing them.

1.1 Information-Rich Environments

Information-rich environments have been around for a long time. We previously defined them in broad terms as environments consisting of a large number and variety of distributed and heterogeneous information sources [5]. The associated applications are varied. They involve the purely informational ones, such as database access, information malls, workflow management, electronic commerce, and virtual enterprises. They also include the information component of physical applications, such as distributed sensing, manufacturing, transportation, energy distribution, and telecommunications. By way of distinction, open environments

- Span enterprise boundaries
- Have components that are heterogeneous in the underlying database management systems used, or the semantics associated with the information stored or manipulated
- Comprise information resources that can be added or removed in a loosely structured manner
- Lack global control of the content of those resources, or how that content may be updated
- Incorporate intricate interdependencies among their components.

Although information-rich environments often involve a significant non-information system component, they are still amenable to specialized multiagent systems.

Cooperative Information Systems are multiagent systems with organizational and database abstractions geared to open environments. Figure 1 shows a CIS schematically. In this figure, we consider an environment consisting of a variety of information resources, coupled with some kind of a semantic directory or ontology for the domain of interest. The semantic directory contains information

about the resources, including any constraints that apply to their joint behavior. Each component of the environment, as well as its human user(s), is modeled as associated with an agent. The agents capture and enforce the requirements of their associated parties. They interact with one another appropriately, and help achieve the necessary robustness and flexibility.

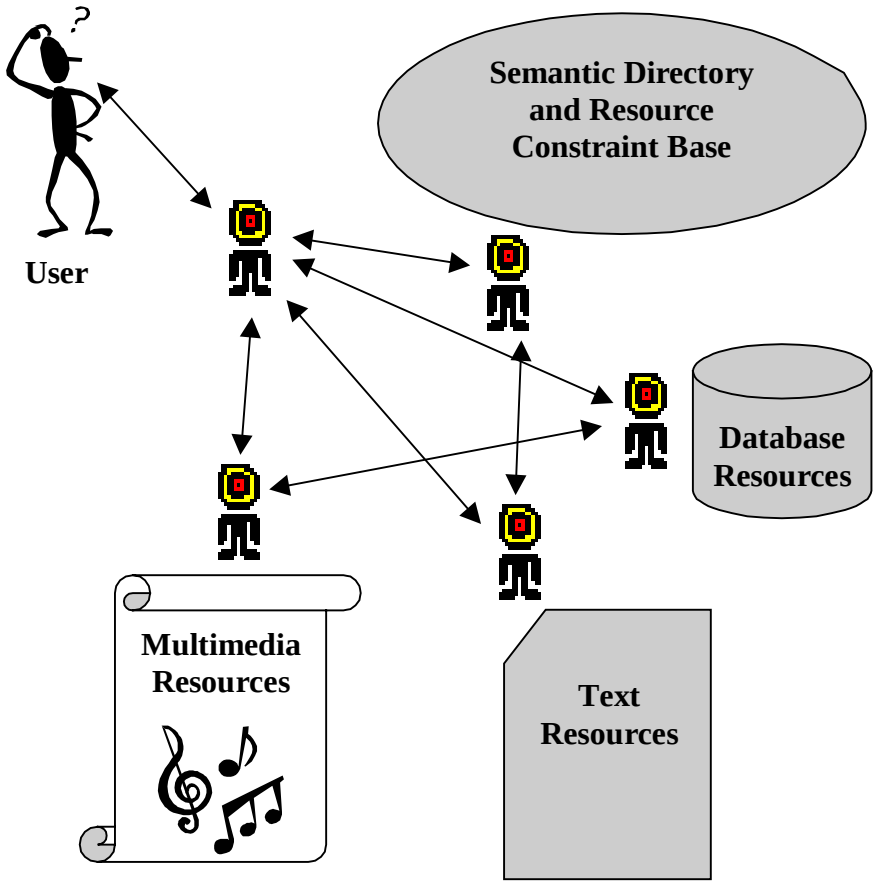

Fig. 1. A schematic view of a cooperative information system, with agents representing the components

Information access involves finding, retrieving, and fusing information from a number of heterogeneous sources. At the level of abstraction that concerns CIS, we are not concerned with network connectivity or the formatting variations of data access languages. Rather, our concern is with the meaning of the information stored. It is possible, and indeed common, that when different databases store information on related topics, each provides a different model of it. The databases might use different terms, e.g., employee or staff to refer

to the same concept. Worse still, they might use the same term to have different meanings. For example, one database may use employee to mean anyone currently on the payroll, whereas another may use employee to mean anyone currently receiving benefits. The former will include assigned contractors; the latter will include retirees. Consequently, merging information meaningfully is nontrivial. The problem is exacerbated by advances in communications infrastructure and competitive pressures, because different companies or divisions of a large company, which previously proceeded independently of one another, are now expected to have some linkage with each other.

The linkages can be thought of as semantic mappings between the application (which consumes or produces information), and the various databases. If the application somehow knows that employee from one database has a certain meaning, it can insert appropriate tests to eliminate the records it does not need. Clearly, this approach would be a nightmare to maintain: the slightest changes in a database would require modifying all the applications that consume its results!

A promising approach is to use mediators [?]. A mediator is a simplified agent that acts on behalf of a set of information resources or applications. Figure 2 shows a mediator architecture. The basic idea is that the mediator is responsible for mapping the resources or applications to the rest of the world. Mediators thus shield the different components of the system from each other. To construct mediators effectively requires some common representation of the meanings of the resources and applications they connect. Such a representation is called an ontology [?], and it is often managed by its own specialized agent.

1.2 Carnot

In order to best understand the work described below, it is useful to review the Carnot project, in which the authors participated. Some of the following ideas had their genesis in the experience gained with Carnot.

A number of innovations were introduced in the Carnot project—these are discussed in [11, 15]. What is most relevant here are the notions of the *semantic* services and the *distribution* services. The semantic services included tools for enterprise modeling and model integration. These tools were used to generate mappings among different resources, modeled in terms of their schemas and terminologies. The mappings were based on a shared ontology for the given domain of interest [14]. The semantics services provided the basis for agent interoperation, letting the agents use models of each other's resources in interacting in a desirable manner. This is similar in some respects to mediators [13].

The distribution services managed logical data access for both retrieval and updates. They included tools such a common interpretive execution environment as well as a distributed communicating agent facility. The distribution services handled the functionality of workflow management, in a low-level distributed computing fashion and in a high-level agent-based fashion.

Thus, Carnot used agents to provide support for relaxed, distributed, concurrent transactions across heterogeneous databases. Carnot was, however, focused

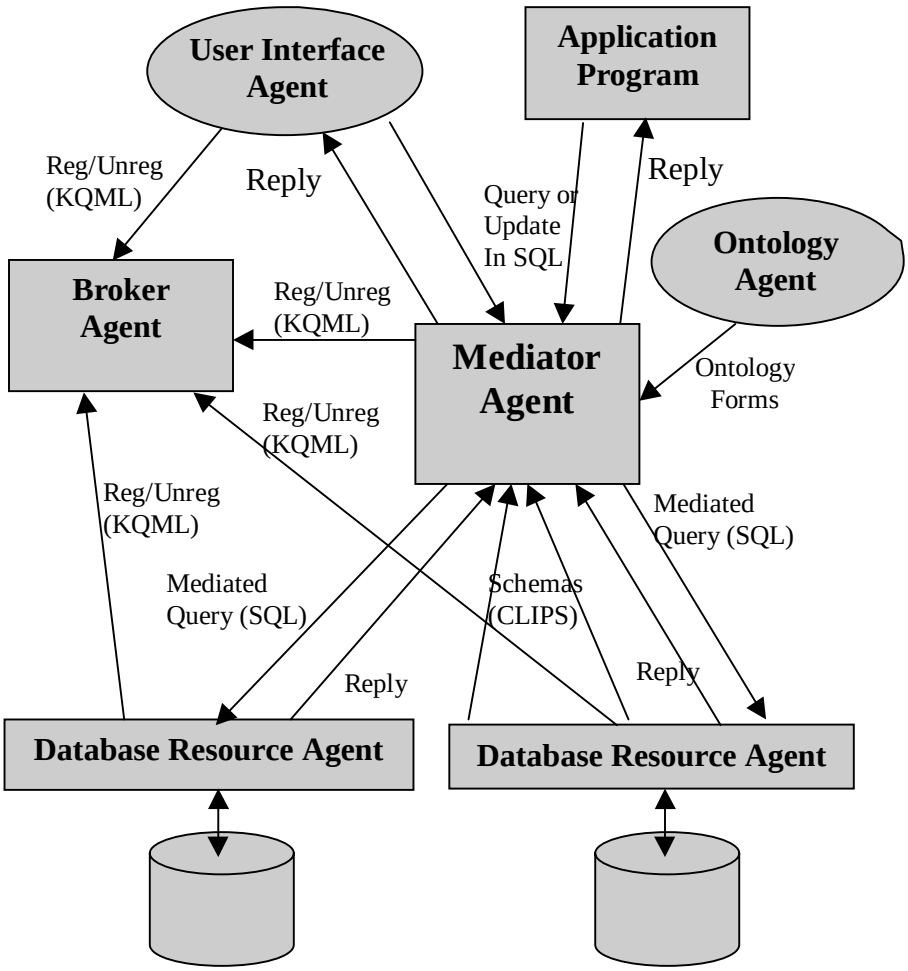

Fig. 2. Agent architecture for a cooperative information system based on a mediator agent

on the problems of enterprise integration within a closed environment, where the component databases were known in advance and fixed.

Of the experiences gained with Carnot, two main observations apply to the enhancements needed to apply cooperative information systems in more general settings. First, although the task of semantic integration was greatly facilitated by the tools developed during Carnot, enhanced integration tools were needed to handle greater varieties of information resources made available by the web. Second, semantic integration was mostly restricted to the static aspects of the information resources, i.e., their schemas and data values, but not for their dynamic aspects, such as their processes and interactions among them. A better understanding of processes, especially as they apply in open environments, is a prerequisite for the integration of the dynamic aspects. There has been some related work in software engineering [2] , but it is mostly geared to single processes, not their interaction.

This paper discusses the results we have obtained along the above two lines of thought.

1.3 MAS Engineering

Although multiagent systems have been known for a number of years and practical applications of them are spreading, they are still being built in a more or less *ad hoc* manner. Prevailing techniques do not support the fundamental properties that make MAS attractive.

Just as for other systems, engineering a multiagent system presupposes the existence of tools and methodologies, which in turn presuppose the existence of suitable representational frameworks and clean theories. As one would expect with MAS being a new area, the state of the art is mixed. Some problems are well-studied or there are clearer inputs from other disciplines—for these, we are seeing useful and practical tools emerging. Other problems are not as well-studied, and the inputs from related disciplines are not applicable to MAS needs, and for these the theories are still begin explored. The approaches described below can be seen as one of each kind.

1.4 Interaction-Oriented Programming

Accordingly, we have been pursuing a research program termed *Interaction-Oriented Programming (IOP)* to develop and study primitives for the specification of systems of agents and constraints on their behavior. These primitives include societies, the roles agents may play in them, what capabilities and commitments require and what authorities they grant. Agents can autonomously instantiate abstract societies by adopting roles in them. The creation, operation, and dissolution of societies are achieved by agents acting autonomously, but satisfying their commitments. A commitment can be canceled, provided the agent then satisfies the metacommitments applying to its cancelation.

The representations for IOP must support several functionalities, which typically exist informally, and are either effected by humans in some unprincipled

way, are hard-coded in applications, or are buried in operating procedures and manuals. Information typically exists in data stores, or in the environment or with interacting entities. Existing approaches do not model the interactive aspects of the above. The IOP contribution is that it

- enhances and formalizes ideas from different disciplines
- separates them out in an explicit conceptual metamodel to use as a basis for programming and for programming methodologies
- makes them programmable

1.5 Organization

Section 2 describes some components of an architecture of a multiagent system for information-rich environments. Section 3 describes the enhancements necessary to apply semantic integration to an open environment. Section 4 introduces a form of commitments that is suited to multiagent systems, shows how they can be operated on, and used to specify social policies. Section 5 applies these notions to formalize applications in electronic commerce and virtual enterprises. Section 6 concludes with a discussion of future directions.

2 Architecture

It is a sign of their maturity that cooperative information systems are beginning to evolve a standard set of agent types. The resultant architecture renders development and deployment of CISs much easier, and essentially raises the abstraction level at which CISs can be described. Some of these are

User agents, which have the following characteristics:
- Contain mechanisms to select an ontology
- Support a variety of interchangeable user interfaces, such as query forms, graphical query tools, menu-driven query builders, and query languages
- Support a variety of interchangeable result browsers and visualization tools
- Maintain models of other agents
- Provide access to other information resources, such as data analysis tools, workflows, and concept learning tools.

Broker agents implement a "yellow pages" and "white pages" directory service for locating appropriate agents with appropriate capabilities. Brokers manage a namespace service, and may have the ability to store and forward messages, and locate message recipients. Broker agents also function as communication aides, by managing communications among the various agents, databases, and application programs in the environment.

Resource agents come in a variety of common types, depending on which resource they are representing, and provide the following capabilities:

- Wrappers implement common communication protocols and translate into and from local access languages. For example, a local data-manipulation language might be SQL for relational databases or OSQL for object-oriented databases.
- SQL database agents manage specific information resources
- Data analysis agents apply machine learning techniques to form logical concepts from data or use statistical techniques to peform data mining
- Resource agents apply the mappings that relate each information resource to a common context to perform a translation of message semantics. At most n sets of mappings and n resource agents are needed for interoperation among n resources and applications, as opposed to n(n-1) mappings that would be needed for direct pairwise interactions among n resources without agents (see Figure 3).

Execution agents, which might be implemented as rule-based knowledge systems, e.g., in CLIPS, are employed to

- Supervise query execution
- Operate as script-based agents to support scenario-based analyses
- Execute workflows, which might extend over the web and might be expressed in a format such as the one specified by the Workflow Management Coalition.

Mediators are specialized execution agents, which

- Determine which resources might have relevant information using help from brokers
- Decompose queries to be handled by multiple agents
- Combine the partial responses obtained from multiple resources
- Translate between ontologies.

Ontology agents are essential for interoperation. They

- Provide a common context as a semantic grounding, which agents can then use to relate their individual terminologies
- Provide (remote) access to multiple ontologies
- Manage the distributed evolution and growth of ontologies. A common context in the form of an ontology or model of the domain can provide such semantic grounding.

Most agent-based information systems incorporate one or more agents of the above types.

3 Semantic Integration and Processing in the Large

A major task for the agents in a cooperative information system is to reconcile the varied semantics of the mostly autonomous resources in the CIS. A focus of our research in CIS is the development of tools for constructing and browsing the ontologies that serve as a basis for semantic reconciliation.

The information available in modern networked environments is no longer just simple text, but now includes multimedia, forms, structured data, and executable code—it has become much more complex than before. As a result, old

methods for manipulating information sources are no longer efficient or even appropriate. Surprisingly, structured data has become more difficult to find and retrieve than unstructured text, because keyword searches over previously indexed documents, which work well for text, are unsuitable for data. Data retrieval requires schemas, which are often unavailable, incomplete, or incomprehensible. Mechanisms are needed that allow efficient querying on diverse information sources that support structured as well as unstructured information.

In such complex and heterogeneous environments, ontologies appear to be well suited for not only organizing new information, but also managing the storage and retrieval of existing information. An ontology is a model of some portion of the world and is described by defining a set of representational terms. In an ontology, definitions associate the names of entities in a universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. For information systems, or for the Internet, ontologies can be used to organize keywords and database concepts by capturing the semantic relationships among the keywords or among the tables and fields in a database. The semantic relationships provide users with an abstract view of an information space for their domain of interest. Ontologies are suitable for graphical representation, and can be scaled and viewed at various levels of abstraction, thereby making them suitable for large information spaces.

As an example of the tools we have been developing for exploiting the benefits of ontologies is the Java Ontology Editor, JOE [**?**]. JOE provides a graphical user interface for users to (1) browse and edit ontologies, or (2) construct queries based on the ontologies.

By being written as a collection of Java applets, JOE can be accessed from any Java-compatible browser, that the same ontology can be simultaneously viewed and edited by more than one user. This group-editing feature has many advantages. It saves storage space since several users can work on copies of a single original ontology. It also eliminates the problem of keeping different copies of the same ontology up to date since only one correct version is saved. At the same time, experts from different fields can jointly build an ontology over a length of time or, if desired, merge various small ontologies to create a single encompassing ontology. This feature is very desirable in large enterprises.

Figure 3 shows JOE in its ontology browsing mode, where it is applying one of its abstraction mechanisms (a magnifier) to help users view a large ontology. Figure 4 shows JOE executing in its editor mode, where users can modify an ontology.

The query mode of JOE is shown in Figure 5. In this mode, users can construct queries by setting constraints on displayed attributes. The constructed query will be shown on a separate window to the right of the main window. JOE will internally translate the graphical query to an SQL statement as the user builds the query. The user can simply submit the query by choosing the "submit" option in the "Query" menu and the results will be displayed in a third

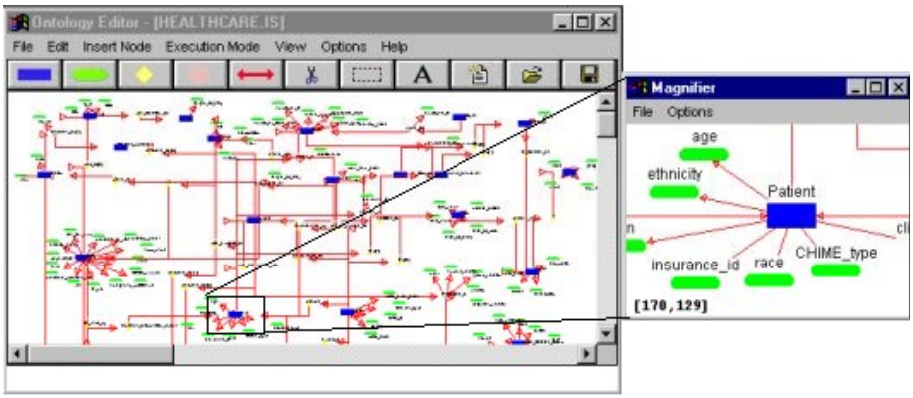

Fig. 3. JOE displaying an entire ontology within its main window and a magnified view of the selected area in the window on the right

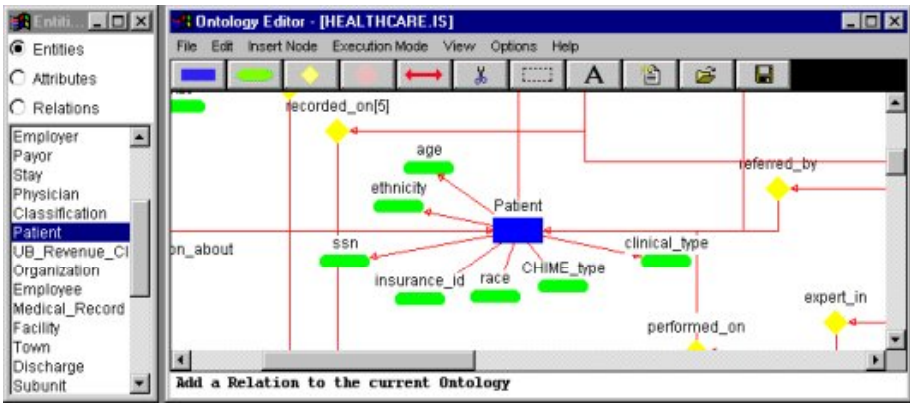

Fig. 4. JOE executing in the editor mode

window. JOE also provides an editor where the user, if he or she is an expert in SQL, can directly modify or type in a new SQL statement for execution.

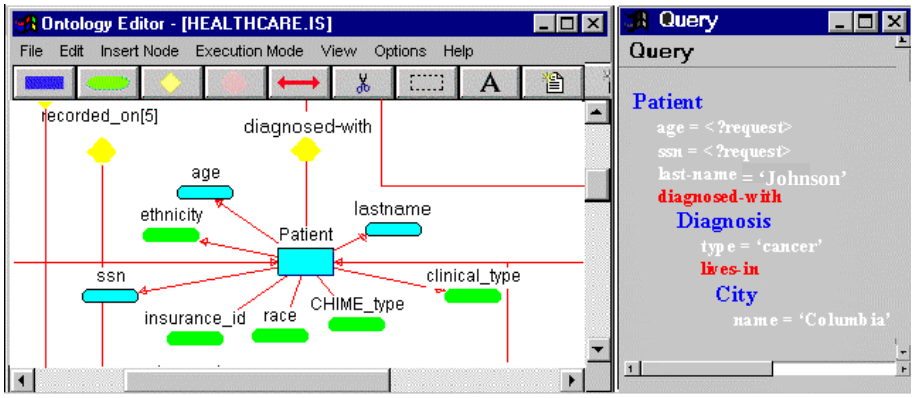

Fig. 5. JOE executing in the query mode, with the partial query "Get the social security numbers (ssn) and the ages of all the Patients whose lastname is 'Johnson' and who were diagnosed-with a Diagnosis named 'cancer' and who live in a City named 'Columbia"'

Now, how can such an ontology be used to facilitate interoperation? It can provide a shared virtual world in which software agents can ground their beliefs and actions. When people talk, they rely on the fact that they live in the same physical world. We know, for example, that a 777 is a type of airliner that can carry passengers to their destination. When agents talk, the only world they share is one consisting of bits and bytes—not a very interesting subject of discussion! An ontology gives the agents a richer and more useful domain of discourse.

Now, suppose our agents have access to an ontology for travel, with concepts such as airplanes and destinations, and suppose that one agent tells another about a flight on a 777. Suppose further that the concept "777" is not a part of that agent's ontology. How could this agent understand the other? The first agent could explain that a "777" is a kind of airplane, which is a concept in the travel ontology. The second agent would then know the general characteristics of a 777. This is illustrated in Figure 6.

4 Commitments

We now turn our attention to the next higher layer of IOP, which deals with commitments among agents, especially as the commitments relate to the social and organizational structure of a multiagent system.

The notion of commitments is familiar from databases. However, in databases, commitments correspond to a value being declared and are identified with the

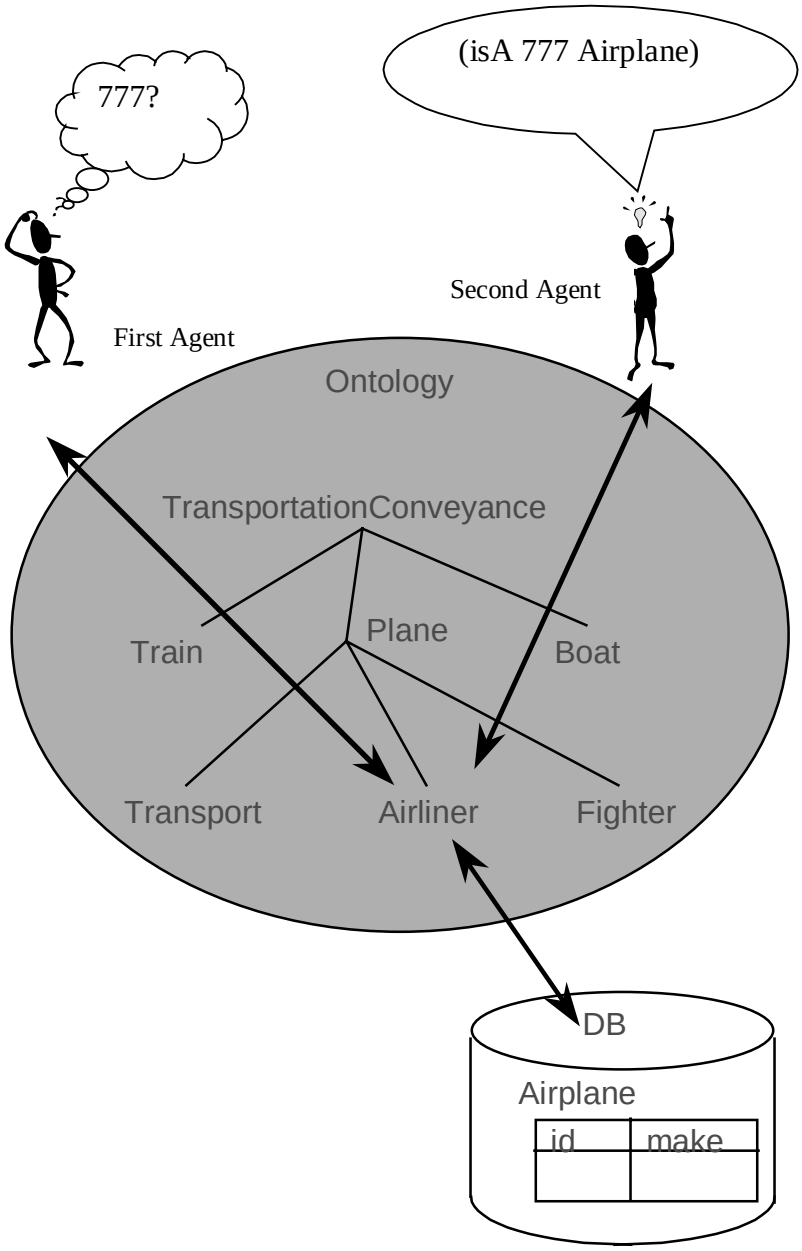

Fig. 6. Agents using an ontology to reconcile their semantics

successful termination of a transaction. When a transaction terminates successfully, it commits, but it is not around any more to modify its commitments. Thus the commitments are rigid and irrevocable. If the data value committed by one transaction must be modified a separate, logically independent transaction must be executed to commit the modified value. Traditional commitments presuppose that different computations are fully isolated and that locks can be held long enough that the atomicity of distributed computations can be assured.

Although suitable for traditional data processing, the above reasons cause traditional commitments to be highly undesirable for modern applications such as electronic commerce and virtual enterprises, where autonomous entities must carry out prolonged interactions with one another [9].

Commitments reflect an inherent tension between predictability and flexibility. By having commitments, agents become easier to deal with. Also, the desired commitments serve as a sort of requirements on the construction of the agents who meet those commitments. However, commitments reduce the options available to an agent.

4.1 Commitments Formalized

We propose an alternative characterization of commitments that is better suited to agents and multiagent systems. In our formulation the commitments are directed to specific parties in a specific context. Thus an agent may not offer the same commitments to every other agent. The context is the multiagent system within which the given agents interact. Sometimes, this multiagent system is termed a *sphere of commitment (SoCom)*. Our approach provides a natural mechanism for commitments to be modified dynamically.

The *debtor* refers to the agent who makes a commitment, and the *creditor* to the agent who receives the commitment. Commitments are formed in a *context*, which is given by the enclosing SoCom (or, ultimately, by society at large). Based on the above intuitions, we motivate the following logical form for commitments.

Definition 1. A commitment $\mathsf{C}(x, y, p, G)$ relates a debtor x, a creditor y, a context G, and a discharge condition p.

4.2 Operations on Commitments

We define the following operations on commitments.

O1. *Create* instantiates a commitment; it is typically performed as a consequence of an agent adopting a role or by exercising a social policy (explained below).
O2. *Discharge* satisfies the commitment; it is performed by he debtor concurrently with the actions that lead to the given condition being satisfied.
O3. *Cancel* revokes the commitment. It can be performed by the debtor.

O4. *Release* essentially eliminates the commitment. This is distinguished from both *discharge* and *cancel*, because *release* does not mean success or failure of the given commitment, although it lets the debtor off the hook. The *release* action may be performed by the context or the creditor of the given commitment.
O5. *Delegate* shifts the role of debtor to another agent within the same context, and can be performed by the new debtor or the context.
O6. *Assign* transfers a commitment to another creditor within the same context, and can be performed by the present creditor or the context.

Through an abuse of notation, we write the above operations also as propositions, indicating their successful execution. We define some additional operations and propositions corresponding to important speech acts. These include $notify$ and $authorize$. $notify(x, y, q)$ mean that x notifies y of q, and $authorize(x, y, p)$ means that x authorizes y to allow condition p.

4.3 Policies

Social policies are conditional expressions involving commitments and operations on commitments. Policies have a computational significance, which is that they can help control the execution of operations on commitments, even without explicit reference to the context. It is their locality that makes policies useful in practice. Agents can commit to social policies just as to other expressions; in this case, the agents' commitments are higher order, and are termed *metacommitments*. An example metacommitment is $cancel(x, \mathsf{C}(x, y, p, G)) \Rightarrow create(x, \mathsf{C}(x, y, q, G))$, which means that x can cancel his commitment for p if instead he adopts a commitment for q (for suitable p and q).

4.4 Applying Commitments

We envisage the following way to apply commitments. Initially, abstract SoComs are defined in terms of their *roles*. Each role is associated with the capabilities it requires, the commitments it engenders, and the authorities it creates. The capabilities are the tasks the agent can do, the commitments are what the agent must do, and the authorities are what the agent may do. The commitments, in particular, may be metacommitments. Indeed, they usually are metacommitments, e.g., that the agent will adopt a base commitment upon receiving a request.

At some point, possibly during execution, an agent may decide to enter into a SoCom as a particular role or roles. To do so, he would have to cause the SoCom to be instantiated from the abstract specification. To adopt a role, the agent must have the necessary capabilities, and accept the associated commitments. In doing so, he also obtains the authorities to properly play the role. The agent must then behave according to the commitments. Agents can join a SoCom when configured by humans or during execution: this requires publishing the definition of the abstract SoCom.

5 Designing Commitments

We consider an example in two parts. The first deals with electronic commerce; the second combines in aspects of virtual enterprises [7]. The commitments are designed based on the corresponding roles in human society.

5.1 Electronic Commerce

We first define an abstract SoCom consisting of two roles: *buyer* and *seller*, which require capabilities and commitments about, e.g., the requests they will honor, and the validity of price quotes. To adopt these roles, agents must have the capabilities and acquire the commitments. Example 1 involves two individual agents who adopt the roles of *Buyer* and *Seller* to carry out a simple deal.

Example 1. Consider a situation involving two agents, *Customer* and *Vendor*, with authority over their respective databases. The SoCom manager has an abstract SoCom for buy-sell deals with the roles of *Buyer* and *Seller*. *Buyer*'s capabilities include asking for a price quote and placing an order. *Seller*'s capabilities include responding to price quotes and accepting orders based on checking the inventory locally. *Buyer*'s commitments include paying the quoted price for anything she orders. *Seller*'s commitments include (a) giving price quotes in response to requests and (b) fulfilling orders that he has accepted.

Customer asks the manager to instantiate a deal between her (*Customer*) as *Buyer* and *Vendor* as *Seller*. The manager asks *Vendor* if he would like to join as *Seller*. When *Vendor* agrees, and since both agents have the requisite capabilities, capacities, and resources, the deal is set up.

Customer now wishes to check the price of a valve with a diameter of 21mm. Upon the receipt of the query from *Customer*, *Vendor*—based on its role as *Seller*—offers an appropriate answer. ∎

5.2 Virtual Enterprises

Example 2 considers a more general situation where the role of *Seller* is adopted by an agent who happens to be a Valvano-cum-Hoosier VE—i.e., a SoCom consisting of the hose and valve vendors. Example 3 considers the situation where the Valvano-cum-Hoosier VE detects a problem in the supply of valves for which an order has been placed. The VE automatically meets its commitments by revising the order and notifying the customer.

Now we consider the situation where one or more agents may form a cooperative SoCom or team. For simplicity, we assume that teams have a distinguished agent who handles their external interactions. We refer to this agent as the VE.

Example 2. We now consider two agents with authority over the Valvano and Hoosier databases, respectively. These agents have similar capabilities to the *Seller* of Example 1. They form a VE, called Valvano-cum-Hoosier VE, which can adopt the role of *Seller*. *Buyer* behaves as before and expects *Seller* to behave

according to the buy-sell deal. However, *Seller* is implemented differently, with commitments among its members, which we do not elaborate here. The possible commitments of the Valvano-cum-Hoosier VE include the following.

- The VE will give price quotes to anyone who requests them.
- The VE will refund the purchase price if an order with matching valves and hoses cannot be fulfilled. There are still no refunds if an order for matching valves and hoses can be fulfilled.
- If the VE cannot fulfill an order, it will try to find an alternative order that will satisfy *Customer*'s requirements.

Recall that *val* or *hos* would not take refunds individually. Thus a customer might be saddled with valves for which matching hoses could not be found. However, when dealing with the VE, a customer can get a refund in those situations. ∎

In the above examples, the actions are performed by the constituents of the SoCom. Sometimes, however, it is useful to perform actions at a higher level SoCom. Such actions might be necessary when the actions of the member agents need to be atomically performed or undone.

Example 3. Continuing with Example 2, suppose an order for matching valves and hoses is successfully placed. It turns out later that the valve manufacturer discontinued the model that was ordered, but recommends a substitute. The substitute valve fits different diameter hoses than the original choice. The VE knows that the original order could be satisfied using the new valve and a different set of hoses. The VE can handle this replacement itself and, based on its prior commitment, not charge the customer any extra. The customer does not need to know of the internal exchanges among the members of the VE SoCom. ∎

In the above example, the discontinuation of a valve after an order for it was accepted is a kind of failure that arises after the original interaction had ended. Traditional approaches would be inapplicable in such a situation.

6 Conclusions and Future Work

We described interaction-oriented programming, and outlined some conceptual modeling issues in it. IOP offers some benefits over previous approaches for building multiagent systems. In the spirit of conceptual modeling, IOP focuses on higher-level concepts than the underlying implementations. These concepts provide a superior starting point to the traditional approaches. Specifically,

- coordination, commitment, collaboration are captured as first-class concepts that can be applied directly
- the underlying infrastructure is separated, leading to improved portability.

Fundamentally, conceptual modeling is as good as the methodologies that one may use to build conceptual models. Accordingly, we have been considering

methodologies that may be applicable to IOP. In the above, we gave a sampler of some of our preliminary results. These methodologies are presently being applied by hand, although there is some work afoot to build tools to assist in their application.

There are some important directions for future research. Of special interest to conceptual modeling is the development of richer metamodels than we have at present. A potentially important theme is to identify useful patterns corresponding to the "best practices" in key areas, and incorporating them in our metamodels. An example area would be contracting among autonomous entities, which seems to underlie several of the upcoming open applications. Along with richer metamodels, there is need for a corresponding intuitive semantics. We have made some progress along this direction [10]. One of the themes that should be more intensively addressed is the compositionality of conceptual models. For example, one would like to build separate models for electronic commerce and virtual enterprises, and dynamically compose them to produce a model for a commercially engaged virtual enterprises. Lastly, there is great need for expressive formal tools that support the conceptual models and their semantics.

References

1. Omran A. Bukhres and Ahmed K. Elmagarmid, editors. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications.* Prentice-Hall, 1996.
2. Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.
3. Stan Franklin and Art Graesser. Is it an agent or just a program?: A taxonomy for autonomous agents. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 21–35, 1997.
4. Michael N. Huhns and Munindar P. Singh. The agent test. *IEEE Internet Computing*, 1(5):78–79, October 1997. Instance of the column *Agents on the Web.*
5. Michael N. Huhns and Munindar P. Singh. Agents and multiagent systems: Themes, approaches, and challenges. In *[6]*, chapter 1, pages 1–23. 1997.
6. Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents.* Morgan Kaufmann, San Francisco, 1997.
7. Anuj K. Jain and Munindar P. Singh. Using spheres of commitment to support virtual enterprises. In *Proceedings of the 4th ISPE International Conference on Concurrent Engineering: Research and Applications (CE)*, pages 469–476. International Society for Productivity Enhancements (ISPE), August 1997.
8. Charles J. Petrie, Jr. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6), December 1996.
9. Munindar P. Singh. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, pages 141–155, May 1997.
10. Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 1998. In press.
11. Munindar P. Singh, Philip E. Cannata, Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Kayliang Ong, Amit P. Sheth, Christine Tomlinson, and Darrell Woelk. The Carnot heterogeneous database project: Implemented applications. *Distributed and Parallel Databases: An International Journal*, 5(2):207–225, April 1997.

12. Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.
13. Gio Wiederhold. Mediators in the architecture of future information systems. In *[6]*, pages 185–196. 1997. (Reprinted from *IEEE Computer, 1992*).
14. Gio Wiederhold and Michael Genesereth. The conceptual basis for mediation services. *IEEE Expert*, 12(5):38–47, September 1997.
15. Darrell Woelk, Philip Cannata, Michael Huhns, Nigel Jacobs, Tomasz Ksiezyk, Greg Lavender, Greg Meredith, Kayliang Ong, Wei-Min Shen, Munindar Singh, and Christine Tomlinson. Carnot prototype. In *[1]*, chapter 18, pages 621–648. 1996.