Intelligent Information Management in a Distributed Environment*

Charles L. Ortiz1 and Marie A. Bienkowski
Information and Telecommunications Sciences Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
(415) 859-5485
Arpanet: bienk@itsd.sri.com

Abstract
In this paper, we discuss the problem of intelligent information management in a distributed environment consisting of a collection of interacting problem solvers. Such environments require applications to be able to adapt to changing circumstances, interoperate to share knowledge and data, and cooperate in joint problem-solving tasks. To support these requirements, we have applied artificial intelligence techniques to design an intelligent information manager (IIM) that provides intelligent data distribution and manipulation based on information it maintains about the state of the network and other nodes, the quality and availability of data, and the priorities of data to be distributed.

Introduction
This paper deals with the problem of intelligent information management in a distributed environment. The type of environment we consider consists of a distributed collection of interacting problem solvers in which new data can enter the system through any subset of problem-solving nodes. When new data arrive they must be made available, in a timely fashion, to other nodes that might need those data. Individual nodes in the system may also choose to cooperate and partition tasks among themselves, and the exchange of information stemming from such interactions must be managed intelligently.

To support timely access to data and internode cooperation, current distributed information systems strive to maintain a consistent distributed database by replication mechanisms [1,2]. Consistency directly supports interoperability since all nodes share an up-to-date view of the world. In order for nodes to be able to cooperate, current distributed systems require that the application programs that constitute the functional problem-solving specialization of each node be jointly developed, so that all of the nodes share the same knowledge and data representations.

In reality, however, consistency and joint development are often unrealistic goals. Future information systems will not always be able to rely on a consistent, shared view of the world for a number of reasons, including time delays and communication constraints stemming from fluctuations in communication services, inoperative nodes brought back into operation resulting in the availability of new information, time constraints imposed on the processing of data, and storage limitations. Furthermore, applications will often be developed independently and will not always share a common language with applications located at other nodes (or even at the same node). Nevertheless, a means must be provided for applications to be able to cooperate with and access data at remote nodes.

To address these problems, we have designed a framework for intelligent information management whose purpose is to support applications and users in managing and using large amounts of distributed data by ensuring that they have access to the timely, high-quality data they need. Making data accessible in a less-than-perfect environment requires supporting interoperability so that applications not jointly developed can share knowledge and data and express data preferences. In addition, it requires supporting adaptability to permit applications to respond to changing circumstances, such as those in which communication constraints prevent access to critical data. Finally, it requires supporting cooperation among nodes in joint problem-solving tasks so that applications can reason about the states of other nodes as well as about the system as a whole (e.g., the state of communications).

The framework we have developed is not instantiated by a single centralized entity in the system, but rather represents a set of capabilities that will allow applications to adapt, interoperate, and cooperate. We refer to this framework, which mediates interactions between an application and the rest of the system (e.g., other applications at other nodes, local and remote databases), as the intelligent information manager (IIM). The IIM has three main functions:

- Maintaining state information that includes a semantic model of local and external databases, dynamic attributes of data, and information about the state of communications.
- Manipulating data based on their quality (a function of their age and source) and availability.
- Using information about the states of other agents, the

---

1 This work was supported by DARPA contract F29601-87-C-0072. APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

2 Current address is: Department of Computer and Information Science, The University of Pennsylvania, Philadelphia, Pennsylvania 19104.
network, and data preferences to assign priorities to data to be distributed and to determine the best source of data in response to a query.

Our approach applies research in artificial intelligence (AI) to the problem of intelligent information management; in particular, it applies research in knowledge representation to the problems of meta-level representations, semantic models, and languages for inter-node interactions. Our approach also draws on work in distributed AI applied to the problem of cooperation among a collection of nodes.

This paper reports on our initial investigations to develop the capabilities needed for intelligent information management. In the following section, we present a problem that motivates the need for such management. Next, we outline the requirements that we have identified for our framework. Then, we present a design for an IIM that satisfies the identified requirements. The complete report [3] presents an extended example and additional details.

Motivating Problem

Figure 1 shows a hypothetical collection of distributed problem-solving nodes. The nodes are able to share information over communication links, and the available bandwidth between any two nodes can vary. Bandwidth is shown by the varying thicknesses of the lines connecting the nodes: a thick line represents good, normal communications while a thin line represents relatively poor communications.

In this environment, new data can enter the system at any node. In Figure 1 we show nodes N3 and N4 receiving new bodies of data, perhaps from sensor reports. A local database is associated with each node.

Node N3 has received new data, called D3, a subset of which—called D3'—is data critical to the operation of node N4. In addition to the object-level data contained in N3's original database (before D3 was received) are meta-level statements that specify a preference or need of D3' data by N4, as well as a description of the state of communications (in this case, including the fact that communication is relatively poor between N3 and N4). Based on these facts, node N3 makes the informed decision to transmit the critical data, D3', to node N4 first.

Node N4 is also receiving a new body of information labeled D4, a subset of which, D4', is data critical to the operation of N2. In this case N4 is aware of the critical nature of D4', but it also realizes that, although communication between N4 and N2 is relatively better than that between N4 and N3, it is still insufficient to support the immediate transmission of D4'. As a consequence, N4 decides to construct a summary of D4' and transmit that instead. Notice that if other, better communication paths between N4 and N2 had been available, N4 would have had to take those alternatives into consideration. Given a better path, the summary may not have been necessary.

Node N2 is faced with a completely different problem. It has just received conflicting reports: fact p from N1 and its negation, ~p, a few moments later, from N3. N2 can resolve this conflict in a number of ways. The default method is to prefer recent data, in which case ~p would be given preference. Depending on the problem domain, another possibility might be for N2 to give preference to an older user report over a more recent sensor report.

Node N5 has become temporarily isolated from the rest of the system. During the period of isolation, it must make the best possible use of the information it has by, for example, interpolating or extrapolating data.

Finally, because of the excellent communication available between N1 and N3, N1 chooses to send a summary of its current state to N3 in order to keep it apprised of changes.

The scenario we have presented has focused only on a single "time slice"; in reality, communications can vary over time, and each node must be able to adapt itself to the changing circumstances.

Information Management Requirements

The functional requirements for adaptability, interoperability, and cooperation translate into a number of technical requirements for the IIM. In order for applications to adapt to changing circumstances they must be able to access a dynamic model of the system. This model must capture "snapshots" of the state of the system (connectivity, bandwidth information, data exchanges between nodes, states of individual nodes—both in terms of data and states of computation—and so forth). These state descriptions must be abstracted as appropriate to the entity requiring the information. This will permit nodes to decide what kind of transmission to make, given the available communications.

The instability of the communications environment implies that there will be situations in which communications will not be available to support distribution or retrieval of all data, so the IIM will be forced to reason about the quality and importance of data in order to prioritize queries and updates, generate summaries, and so forth. It is necessary, then, to be able to represent attributes of data that permit such reasoned decisions to be made. For example, the quality of data is an important attribute, and it is primarily a function of its age and source. Thus, dynamic facts must be indexed by the IIM according to their age and source. Application programs might wish to specify other attributes of data; for example, applications might wish to specify the expected persistence of a fact or how long a history should be maintained for a fact.

When facts are labeled according to their source, age, and other attributes, application programs can construct policies that specify desired data manipulations based on these attributes. For example, to handle multiple reports that translate into a pair of inconsistent facts, an application might define a policy that gives preference to more recent sensor reports. Such a policy might be used by the IIM during data retrieval to deconflict those facts.

To minimize the burden on limited communications resources, nodes should first share only critical data, specified by data preferences contained in profiles; other, less critical data can then be exchanged if there is sufficient bandwidth. In some cases, the IIM may only be able to send partial or incomplete data, in which case the receiving node may have to reconstruct or estimate the state of the sender. Some applications may allow nodes to function with only summarized data and high-level abstractions.

An ability to generate summaries of data and to specify profiles and policies supports function and data survivability. Profiles are updated when data preferences change,
and are distributed across a network so that other nodes can make decisions regarding the prioritization of data for transmission. In this way, critical functions can continue because the critical data necessary for their operation is available. Further, when communication constraints are especially stringent, nodes can generate summaries in order to communicate some abstraction of the information critical to the node that specified the profile. The distribution of policies that represent expected data changes (movements of a vehicle, for example) can also provide a measure of survivability since nodes can distribute them in lieu of periodic updates, which might become impossible because of future communications restrictions.

In order for nodes to specify these profiles and policies, they will need to share a common language in which to express statements and actions. This language must be understandable by other applications at other nodes that might use databases and knowledge bases with different representations. Applications should be able to refer to attributes of an object directly rather than, for example, by reference to a particular slot in a database record.

Finally, in order for nodes to reach data exchange decisions, they will need to reason about the states of knowledge of other nodes and reflect on their own states of knowledge. We do not describe our approach to this issue in this paper; details can be found in another report [3].

Design

The IIM introduces a layer of processing between applications and the database and communications layers. Applications and users make requests for data through the IIM, which sends requests for data to other nodes (if the data are not available locally) by constructing a message for delivery by the communications layer. The communications layer provides raw data, such as traffic volumes and network topology, to the IIM. The IIM also handles data updates made by applications and users by updating a local copy, and then using profiles to decide whether or not to distribute the data.

As described previously, the IIM performs three types of processing: maintaining state information, manipulating data, and distributing data. Figure 2 shows the various components of the IIM. An instantiation of this framework is at each node. Modifications to state information are made via the IIM interface, which provides access to (1) an acquisition component for interactively modifying both a semantic model of conceptual entities and access relations to capture changes made to the structure of the database, (2) a network state generator (NSG) that translates raw communications information into an abstract form suitable for higher-level reasoning mechanisms, and (3) a data management interface through which profiles and policies can be changed. The state information comprises data of various kinds:

- Object-level data, which is the information stored in a distributed database just as it is in current systems.
- Meta-level data, which captures data attributes such as how old a piece of data is, or whether it originates from user reports, default assumptions, derived data, or sensor reports.
- A semantic model of system databases, partitioned according to the location of data (local or external).
- Access relations, which provide translation mechanisms between concepts represented in the semantic model with relations in the database.
- Descriptions of the states of knowledge of other nodes in the system.
- The state of communications and other resources.
The data manager (DM) responds to requests for information or updates from applications. It uses a knowledge base of policies and profiles to plan data management actions (for example, to deconflict data or to prioritize the transmission of reports). The DM also maintains a record of the actions and changes it has performed. In the sections that follow we describe each of these components in more detail.

The IIM Interface

Concept Acquisition

Because databases evolve, it should be possible for a user to alter the semantic model as the database schema changes, without having to be an expert in the mechanics of the semantic and data representations. Our approach to this problem is to apply research in transportable natural-language interfaces [4]. In that work, natural-language understanding systems are transported from one domain to another by modifying the conceptual schema that links lexical entries to database entries. Using this technology, a user (such as a database administrator) can, through an interactive session with the IIM’s concept acquisition component, modify a conceptual hierarchy, adding or deleting concepts, and link objects in the hierarchy to database items.

The acquisition component first prompts the user for the name of a relation and its arguments. The new relation is added to the semantic model by entering the path to a related concept. The related concept may be an existing concept or a new one that is related to existing concepts through inheritance (isa) links. The acquisition component will also request the names of the arguments in the relation, so that these attributes can be made explicit through access relations (described shortly).

Network State Generator

The NSG serves as the interface between the IIM and the network (including network management processes such as resource monitors [5]). The NSG is designed to convert raw network parameter values (such as traffic volumes, hop counts, signal strengths, retransmission counts, error rates, and queue lengths) into a representation that expresses the state of the network in terms of higher-level abstractions (such as the degree of vulnerability of a node or link to outside interference) [6]. Each abstract network characteristic also has a predicted counterpart (for example, predicted congestion or predicted throughput). These abstractions are stored in the communications database. The NSG contains experiential knowledge obtained from network experts and used to interpret and analyze the raw network data and produce these abstractions.

The NSG consists of two components: knowledge sources (KSs), each of which describes how a given raw parameter value is analyzed to form an opinion about the value of a specified network characteristic, and frames, which specify the details regarding the KSs that must be executed (and hence, the set of raw parameters that must be analyzed) to form a complete opinion about the value of each characteristic. Evidential reasoning [7] provides a formalism whereby the opinions expressed by these separate KSs regarding a given network characteristic may be combined to form a single body of knowledge. The net result is a set of assertions about network characteristics that can be used by the DM as it plans data distribution actions.

State Information

An important requirement for intelligent information management is an information management language (IML) that is sufficiently expressive to characterize the state of a distributed system abstractly and to refer to data objects...
and their dynamic attributes. The IML will serve as (1) the common language of the individual nodes that periodically share their views of the state of the system, (2) a common language for database access to an existing relational database, (3) a way to refer to data objects and their attributes, and (4) a means to express profiles and policies.

We have chosen the logic programming language Prolog in our initial design efforts. In a declarative language like Prolog, attributes can be introduced and associated with existing data objects. New, complex relations can be defined in Prolog that reference existing database relations. These new relations can then, for example, be monitored for changes. In this way, existing databases need not be restructured every time a new application wishes to define new attributes.

Prolog is attractive as an implementation language because it supports bi-level reasoning: that is, it can be used to answer database queries while at the same time reasoning about the actual data and their attributes at the meta-level [6].

Semantic Model

A semantic model of data provides a structure independent way of referencing data and maintaining the integrity of data [9,10]. Structure independence directly supports the requirement for interoperability discussed earlier. Instead of an application having to know the exact schema of a database (for example, that the location of a truck is represented in the “Loc” column in a relation called TRK), it can query the database for a particular attribute of an object through an access relation defined in terms of the shared, application-independent, semantic model. (A typical access relation may look like location(truck, LOC) :- trk(truck, _, Loc, _).) Using the semantic model, an application can also express policies and profiles that can be distributed to other nodes that can understand them via the semantic model.

Another advantage of the semantic model is that more general concepts can be expressed using the ISA hierarchies. For example, trucks, jeeps, and tanks are all vehicles, so an application at a particular node could broadcast a compact message stating its interest in “reports of vehicles in a particular location” instead of having to request “reports on trucks, jeeps, and tanks.” This sort of economy of expression is very valuable in situations characterized by communication constraints.

Since nodes can be functionally specialized, the IIM will be able to capture the notion of “what sorts of things node X knows about” in the semantic model by partitioning it according to the areas of responsibility of nodes. This is useful for a node in determining the best source of information.

Data Attributes

We have already noted the importance of augmenting the ontology of existing distributed databases by adding new objects that are associated to old via, for example, ISA links. It is very likely, however, that applications will require further augmentations. In this subsection we describe a preliminary set of data attributes that applications and the IIM might need. Note that because of the modular design, each application or user can introduce new concepts or attributes through the Prolog interface without altering the structure of the existing database.

We adopt a syntactic treatment to the problem of specifying these dynamic data attributes by using meta-predicates in Prolog. Facts that were propositions in our database become terms in predicate arguments. These predicates capture the attributes of the facts. Below is a list of the sorts of meta-predicates we have in mind:

- source(Fact, Source) specifies the source of the referenced fact, for example, raw (e.g., from a sensor), user-supplied, derived, or default.

- holds(Fact, T) specifies that Fact was true at time T. Two inconsistent facts holds(fact, t1) and holds(~fact, t2) can thereby be distinguished in this way.

- persistence(Fact, Length). Processes might wish to specify the anticipated persistence of data (any data token that exceeds its lifetime might be subjected to a garbage collection routine—depending on the appropriate history policies). Persistence might be specified in terms of a monotonically decreasing function.

- history(Fact, FactList). Applications may wish to define history policies that allow the system to have a limited form of memory by maintaining a history of a particular fact (say the movement of an object) in this form. The history must be maintained at the appropriate level of abstraction. For example, policies could be used to specify how much history is maintained and a means for creating new abstractions of the data (for example, moves predicates from changes in location).

- eff(Fact, Confidence). Based on policies for deconfliction and the like, certainty factors would be associated with a particular fact. Note that the semantics here is left open for the application.

Knowledge States

The IIM needs to represent the states of knowledge of other agents. In AI, two primary approaches have been developed for representing states of knowledge. One is a semantic, possible worlds approach [11,12]; the other is more syntactic [13]. The semantics of the syntactic approach associates a set of beliefs (i.e., a theory) with each agent and then K_i,fact is true just in case fact is in i’s set of beliefs. In our domain, we would like to make statements of the form Knodereceived(nodex, report(objects, area)), intended to mean that nodea knows that nodeb has received reports on objects entering some area. One would also like to express embedding of knowledge, such as what nodea knows about what nodeb knows about, as well as temporally indexed statements. The latter has been investigated recently [14]. In another report, [3] we discuss these issues a bit further, however, because this is an area of considerable ongoing research, we view implementations that apply these epistemic notions to be far-term.
Data Management

The DM's task is to compute data manipulation and data distribution actions based on the available state information and the policies and profiles in the data management knowledge bases. The DM provides the control necessary for deciding which inferences to draw based on the policy and profile axioms. The control provided by the DM can be as simple as the serial processing of policies and profiles. For example, if there were a profile that indicated that a particular node had a need for a certain type of information, then, at this level of reasoning, the DM would decide to assign highest priority to the distribution of that type of information to that particular node. More complex processing would involve decisions over competing policies or profiles, or the use of axioms containing statements regarding the states of knowledge of the various entities that constitute the system, as well as the state of communications.

The DM is responsible for processing queries and updates. It prioritizes requests for information based on policies and attempts to satisfy the request with the highest priority. When responding to a query, it first accesses the local object-level data. If the data are available locally, it returns with an answer together with an estimate of the quality of the data based on meta-level information (for example, how old the data were, where they came from, etc.). Should the requested data be unavailable locally (either because they are not known, are out of date, or are in conflict with other data), then the data manager applies the appropriate policy to either retrieve the data from a remote node (if communications permit) or to infer a best estimate of the data. Data updates are performed locally by the DM, then distributed according to profiles. Updates are checked to make sure that they meet the integrity constraints specified by the semantic model.

The DM can conserve scarce communication resources and support survivability by sending a new policy (specifying, for example, how data are to be deconflicted, or what data should be monitored), instead of executing the policy first locally and then distributing the results. (The number of results may be larger than the "size" of the policy. In addition, problems might arise if the node transmitting the results is suddenly cut off.) A similar benefit is gained if a process, wishing to periodically announce default updates, instead chooses to transmit a default policy. By expressing the policy in the language of the semantic model, the node being updated can understand how that update is expressed (in contrast to the alternative of interpreting a message consisting of a table describing how that data is expected to change; the table might not be understandable by the receiving node).

Our initial design efforts for the DM apply the technique known as meta-programming (writing programs that treat other programs as data). A meta-interpreter is simply an interpreter for a language that is written in the language itself. (This is especially simple to do in languages such as Lisp and Prolog, but more difficult to do in languages such as C.) The advantage of a meta-interpreter is that it can be enhanced to extend the capabilities of the base language in order to simplify the expression of complex programs. The meta-interpreter can also be extended to support debugging and to gather information about the execution of programs.

Data Manipulation Policies

Data manipulation policies provide a means for reasoning about data based on their quality, i.e., their age and source. These policies can range from a simple set of axioms to complex modules that perform detailed analysis of data. One advantage of separating policies from the application level is that they can be distributed globally and thereby provide a means for nodes to effect changes to data locally (using those policies). The less preferred alternative is for applications to process data and then distribute the results. This approach fails in the case of severe communication constraints, which prevent the distribution of the computed results and thereby compromise the system's survivability.

The following is a partial list of policies we propose for manipulating data:

- Deconfliction. Deconfliction policies can range from simple ones based on recency of data to more complex ones that take into consideration the source and its reliability.
- Integration. There may be dedicated fusion modules responsible for performing sophisticated data integration.
- Interpolation. Data can be interpolated spatially, temporally, or according to other problem-specific methods.
- Extrapolation. New data can be inferred by reasoning about the change in old data over time. A sample extrapolation policy is one that projects the location of a moving object forward in time on the basis of an existing plan, if recent information on the movement is lacking.

Data Distribution Policies

A wide variety of domain-dependent data distribution policies would need to be expressed in the IML. Because these policies are, for the most part, domain-dependent, the reader is referred to another report [3] for detailed examples. As in the case of data manipulation policies, we can identify a set of important types of policies for data distribution:

- Monitors specify which facts must be monitored for changes; these changes could then be sent to a local or remote application.
- Summary Policies describe how summaries can be generated when full responses to remote queries are not possible because of communication constraints. Examples of data summaries include sending only the important fields of a data request, or sending a total rather than individual subtotals, or, instead of giving the location of every constituent part of a group of objects in response to a query for location information, giving an average.
- Profile Use Policies specify the relative importance among a group of profiles.
- Resource Policies describe the priorities among sets of resources (processing as well as communication).
- Communication Policies specify, under the known constraints, what to say to whom and how to say it.
These policies might make reference to the states of knowledge of the various entities that constitute the system, as well as to the global state of the network.

Summary and Future Work

The approach we have described has the following desirable features:

- It provides a framework for intelligent information management that ensures that applications and users will have the critical information that they need when they need it.
- It makes use of meta-level statements about data, provides a common language with which applications can interact, and permits intelligent reasoning to derive data manipulation and data distribution actions.
- It is a great improvement over traditional solutions that rely on consistent databases: such approaches fail in situations characterized by unstable communications. The proposed approach is also an advance over approaches that place information management responsibilities at the application level: such approaches lack modularity and increase traffic within a network.
- Finally, our design directly supports the long-term goals of interoperability, adaptability, and cooperation among nodes in a distributed system. In short, it supports data and function survivability within the system.

In a recent report [3], we present many useful policies that can be used to manage information intelligently. The work described part of an ongoing effort to develop an IIM within an actual domain by using existing databases and existing applications, developing the necessary semantic models, acquisition component, and meta-level databases together with scenarios with which to test the system.

References