include tr91.refer

TECHNICAL NOTE

ISA: An Architecture to support Sharing and Exchange of Information among Cooperative Archon Agents

AUTHORS

Hamideh Afsarmanesh Frank Tuijnman

Computer Systems Group University of Amsterdam

December 1991

This technical note describes the design and development of an Information Sharing Architecture (ISA) to support agents' cooperation in a distributed, loosley-coupled, multi-agent system. Agents are semi-autonomous, running on one or more real processors, and the control is decentralized. This approach is currently being developed in an ESPRIT (European Strategic Programme for Research and Development in Information Technologies) project P2256, called ARCHON (ARchitecture for Cooperating Heterogeneous ON-line systems). Archon is a framework for embedding a collection of software systems (ISs) in a single cooperative architecture. Each IS (Intelligent System) has the knowledge on different aspects of a complex problem domain. However in general, each IS has only partial knowledge with respect to the entire problem domain, and some overlap of knowledge exists among ISs. In a control environment for example, some ISs may be controlling parts of an external physical system, or they may support and advise human operators who are controlling parts of an external physical system. These cooperating software systems are initially decoupled from each other. This means that ISs may use different local mechanisms to represent their knowledge, they may associate different structures and semantics to the represented information, as well as employing different local control mechanisms. Our research at UvA is focused on addressing the problem of sharing and exchange of information among semi-autonomous cooperative Archon ISs. Within a system of cooperating ISs a large variety of information is manipulated. Some of this information describes the world that the system acts on, some of this information is used to control the cooperation, and some of this information defines the relation between other information and the elements that process information. The problem of information exchange among cooperating ISs is due to the following facts. Typically, each IS independently (locally) defines its own model of information, data structures, and operations to process the information. On the other hand, sharing of information among ISs usually involves large volume of complex structured data. Therefore (without an information management framework), data sharing within Archon community is performed in a completely adhoc manner. For one IS to receive some data from another, it must also receive all the structural information and semantics defined on this data. Also, the entire task of interpretation and purifying the received data to make of suitable for use must be performed at the receiver IS itself, which makes the transfer of information quite costly. Furthermore, consider the case where there is neither a global data exchange mechanism, nor a global information modeling formalism to be used by ISs. Then, the details of the structural information (namely, the information representation, semantics, and operations defined on the information, and the distribution of inforamtion among ISs) must all be known to ISs involved prior to information exchange and is partly included in the application programs and partly in users' minds. As a result, the structural information are redundantly stored within the Archon community of ISs, and the exchange of information becomes quite vulnerable. Another drawback of this redundant storage is the consistency problem resulted when ISs modify their structural information.

Here we have chosen one example situation from solid modeling area to represent the problem of interactions among ISs that apply different representation mechanisms to information. The problem is illustrated by the interactions that a high level robot controller (HLI) IS has with a vision system IS and a CAD system IS for solid modeling. Later in Section 6, where we describe the implementation strategy of ISA, we represent a solution to this problem. The HLI monitors the operation of the robot. In case an exception occurs, which makes it necessary to abort the current robot program, the HLI will try to diagnose the exception and to make a new plan for the robot so that it can continue. One of the capabilities of the diagnosis system of the HLI is to identify unexpected objects. To do so the HLI must direct the camera to the proper position, tell it to produce a description of the object it sees and then the HLI has to compare this information with the information produced by a solid modeling CAD system. The vision system generates an adjacency graph of the edges it has seen. Each edge has length as an attribute, and the adjacency relation has as attribute the angle between the edges. The CAD system produces also a graph of the boundary of an object, but this graph has a structure that is quite different. For example, instead of explicitly representing the angle between edges it will only provide the parameters of the edge equation. Another aspect is that it contains not only the description of the edges, but also explicitly represents the surfaces and the vertices. And finally, apart from the representation issues, both models of the same object are inherently different, because the vision system observes only part of the real-existing object, while the CAD system provides a complete description of what the object should be. Before the HLI can compare these two representations it needs to extract from the CAD model the graph which only contains the edges of the object, and has the same type of attributes as the graph generated by the vision system. In other words it has to create a single representation in which both descriptions can be expressed. After the conversion, remaining differences between the two descriptions have application specific causes, in this case occlusion which makes it impossible for the vision system to see all edges at once, and disturbances due to observation errors. The problem of the HLI in using the descriptions of the objects provided by the CAD system and the vision system is that although they both provide a similar description of the same aspect of the world (the geometry of an object), this description is completely different, both with respect to semantics and the representation of the world. The main goal of embedding independent semi-autonomous ISs within one system is to obtain a single environment composed of loosely-coupled software systems that cooperate to achieve better performance in handling tasks and responding to users requests. Archon will have to provide a universe of discourse to support the cooperation among these ISs. Each IS represents some pieces of information (results) that they are able to produce, contains a description of the information they need to receive (this information is produced by other ISs) and mechanisms to receive that information, and has its own local control knowledge. In order to support the cooperation among ISs, the Archon architecture introduces the Archon layer for each IS. Archon system is then defined as a collection of interrelated Agents, where each agent is an IS together with its Archon layer.

Archon agents use heterogeneous models to represent their information which in turn complicates the exchange of information. A first step to deal with this problem is to develop an appropriate formalism in which the information that an agent produces, or needs to receive, can be described. For example, an object-oriented framework can be considered in which the information are modeled by objects and which provides functions to query and update the information. This approach has the advantage that no distinction is made between the information that is actually stored as data and the information that can be calculated with the available data. For example in the case of an edge graph of objects it is of no concern to the user whether the angle between two edges is represented explicitly in the data structure or whether it can be calculated on the basis of the edge equations. An object-oriented framework by itself however, does not suffice to create a common denominator to describe IS's information. It is necessary to support data models and higher level representation mechanisms in which the information of agents can be expressed more concisely. Modeling constructs supported by the framework must be general enough to represent the information common to the Archon application domain and at the same time capable of representing the specific information of agents. These modeling constructs can then be the building blocks of a global formalism and framework in which agents are described and can exchange information among each other. So that, for example, the operations on a graph that represents a road map, are the same as the operations on a graph in a different system that represents an electricity network. Such a global formalism and framework can be included within and supported by the Archon layer. The Archon layer can then offer support to cope with the representational differences between semantically compatible structures defined in different ISs. A major advantage of this global formalism is that the information structures that reoccur in many systems, can be made to look identical at the Archon layer. This document represents the detailed design of ISA and describes the implementation of ISA through the development of AIM (Agent Information Management) modules for ISs. AIM modules support the complex information-representation and information-manipulation of the knowledge being shared and exchanged by agents, as well as providing a distributed information exchange mechanism to support the ISs' interactions. An AIM module is developed for each IS and is included in its Archon layer. Therefore, agents can communicate and exchange inforamtion through their Archon layers. In this document the specification of AIM module and its components namely, the 3DIS data model, the 3DIS/ISL query and update language, and the 3DIS/DIA distributed information access framework are presented. For each of the three components, first a specification of the component is presetned and then the required extensions to the component to make it more suitable within the Archon system is described. 3DIS Afsa89a is the object-oriented information modeling formalism, used for AIM. The 3DIS schemas generated for ISs can describe the application environments and define the structures and semantics associated with the information produced and exchanged by agents. Data modeling constructs supported by the 3DIS can be used as a base on top of which the structures and the operations specific to application environments can be defined. This however, does not have to be done in one step. Instead a hierarchy of concepts can be developed where the concepts closer to the ROOT (of the 3DIS-kernel) define more general structures common to many application environments. Concepts further away from the ROOT will support increasingly narrow application domains. The description covered in this document is essentially limited to the 3DIS-kernel (the Root and the first branches of this hierarchy). The 3DIS-kernel introduces several primitive notions that are common to all environments in which data models are to be defined and consists of the basic structures used by applications. The 3DIS modeling formalism is extensible in the sence that it can support the definition of domain specific Abstract Data Types (ADTs), Tuij89 Meij88 as extensions to the 3DIS schemas. Afsa90a An information-retrieval and information-manipulation language, called the 3DIS/ISL is defined for the 3DIS. Afsa89a This language supports the local access and modification of agents' information, and is used by 3DIS/DIA to support the remote access and sharing of information among agents. A BNF-like definition of this language and a brief description of its commands is covered in this document. The AIM module also provides a distributed information sharing formalism, called 3DIS/DIA, which is defined on top of and uses the 3DIS model and the 3DIS/ISL language to support both the exchange of information and the autonomy of agents involved. A description of this distributed information sharing framework is provided in this document.

In this section we describe the design of ISA (Information Sharing Architecture) to support the exchange and sharing of information among Archon agents. The main emphasis here is on ‘static’ information, in other words the issue "when what information is available" is not discussed here. Exchange of information among agents can be organized in several different ways. One way is to have for every two agents that share information a small number of functions that they can use to access each others information. This mechanism is quite inefficient, since on one hand it requires a complex negotiation between every two agents in the Archon community to agree on the functions and on the other hand, the functions generated themselves are often redundant, data-dependent, and access-dependent. A second way is to let each agent provide a list containing a fixed number of functions to access the information it wishes to share with other agents. This technique also has several disadvantages, for example there must be an a priori design of the access method for the information inside each agent. Further, the designed access method must be capable to provide all the information needed by other agents. Another disadvantage of this technique is the amount of post-processing each agent must perform on the received information to make it compatible with its local information. The third way is to define a global information access/sharing mechanism that uses a global data model to represent the information of each agent, and provides a global language (such as a relational, functional, logical, graph-based, etc.) defined on the global model to support the agents’ interactions. This technique which is the approach taken for ISA provides a higher level mechanism of information exchange than the other two. In fact, the availability of a global formalism and framework forms a non-biased starting point for designing the structure of information exchange and agents’ cooperation, for which it provides a minimal set of constraints. The first two approaches discussesd above aim at resolving the following problems: naming of objects and relationships can be different among agents. information distribution over agents (where is what information) is unclear. there is the need for specific procedures for accessing the information in each agent. The third technique however, approaches and resolves a wider spectrum of communication problems among agents: It establishes a common framework among the agents to define information structures and for manipulating them. It is more flexible, so no change is needed to be made to an agent if other agents need to access different parts of its information, or if they need to access its information in a different way. It supports the philosophy of loosely-coupled systems where we don’t want to fix agents’ interaction (in this case the access of information), in an a priori manner. It supports the efficiency. If an agent is interested in a small subset of information (e.g. a few elements out of a large set), it is much better to perform the selection at the agent’s site where the set is stored, rather than transfer all the data (most of which is not needed), and let the receiving agent perform the selection. Also, where no simple query language is available, one would have to introduce new message types for all commonly used queries. In any case for the user interface a tool (language) has to be developed to allow the user define which information he is interested to receive and in which information he is not interested. So it is simply a matter of making this interface also available to the communication between the agents. This is as if we consider an agent as another user. In addition, another reason to explore this approach, is the fact that the exchange of information with a complex structure, as it occurs within the robotics testbed, requires a proper model. Much of the information in the Archon system does not consist of simple atomic facts but of complex structures of relations and facts. The way these relations are defined depends on the usage that an agent makes of these relations. Typically, two agents that use the same information, especially if they are developed separately, will define the structure differently. This leads to problems, first in the exchange of information, where conversion is needed, and second in maintaining the consistency between the different structures when they are modified. Also, it facilitates the design of each agent if the agent is free to choose the representation that is best suited for its needs. ISA serves to provide a framework for interconnected agents in which they can maintain their autonomy in modeling and sharing the information, yet it supports mechanisms for a substantial degree of complex information sharing, and while minimizing the central authority. Therefore, the Information Sharing Architecture (ISA) presnted here is an approach to the coordinated sharing and interchange of computerized information among autonomous, loosely-coupled agents. Applying the principles of Archon (loosely-coupled, semi-autonomous agents) to ISA, we can state the following requirements: Agents cannot be forced to provide information to other agents. Therefore, the role of central autority is replaced by cooperative activities among agents. Agents can determine their own view of existing data. (There is no global schema (describing the information produced by all agents, rather, each agent builds its own schema). To realize this approach to information exchange by ISA, we need to develop the following components: A global information modeling formalism in which an agent can describe both the information it wishes to share with the community and the information it wishes to receive from the community. A global query and update language to access the available information, to be used by all agents in the community. A global method by which agents can know about the information of other agents and by which they can access that information remotely. A main requirement for the global information modeling formalism is that it should be possible to easily model the complex information of agents and express the concepts that agents communicate among each other. A main requirement for the global language is to at least support the basic primitive query and update operations as well as being capable to support higher level operations specific to certain ISs. In the high level description of ISA, some preliminary choices are made for its components. It should be emphasized that no exhaustive search of ‘all’ possibilities to find the ultimate ‘best’ answer has been carried out. Rather, the goal has been to find solutions that are adequate and well tuned to each other, but sufficiently simple to be feasible within the scope of the Archon project, and which are roughly as good as (a decision that was based on our expertise in the area) the other alternatives. The global information modeling formalism used for ISA is the 3DIS (3 Dimensional Information Space) an extensible object-oriented information management model. The global query and update language used for ISA is the 3DIS/ISL (3DIS/Information

Sub-Language) developed on top of the 3DIS model. The 3DIS formalism and the 3DIS/ISL language described here has been developed by Hamideh Afsarmanesh and is documented in ‘‘The 3DIS: An Extensible Object-oriented Information Management Environment,’’ by H. Afsarmanesh and D. McLeod, in ACM Transactions on Information Systems, October 1989. The design of a global distributed information access method for ISA, called 3DIS/DIA (3DIS/Distributed Information Access), is based on the Federated information architecture. Heim85 IEEE87 The federated information architecture, developed by Dennis Heimbinger, Heim85 was originally designed for integrating local databases in an office environment, while allowing complete local control over each of them. The 3DIS/DIA formalism developed in ISA provides mechanisms for sharing data and operations on data, and for combining information from several agents to facilitate the coordination of activities among autonomous agents.

The information modeling formalism chosen for ISA is the 3DIS(3 Dimensional Information Space). Afsa89a The 3DIS is a simple but extensible object-oriented information model. It is defined in terms of objects (a subset of which are mapping-objects) and relationships defined among objects. Objects are referred to by a unique designator, or by their value if they are atomic objects (e.g. strings, numbers). Relationships are simple triples of the form "(domain-object, mapping-object, range-object)". Some basic reasons for choosing the 3DIS are: 3DIS is simple, and a simple implementation can support it, so no major implementation effort is required. 3DIS has successfully been used for engineering and design applications. Afsa90b Tuij90 Afsa89b Afsa86a Afsa85c This makes it more likely that it will be easy to express the concepts of the application domain of Archon. 3DIS integrates well with the architecture for information sharing, which is used to access the remote information. A 3DIS model of application is a collection of inter-related objects, where an object represents any identifiable piece of information, of arbitrary kind and level of abstraction. The 3DIS unifies the view and treatment of all kinds of information as objects. The uniform treatment applies to both data and the description and classification of data (also called meta-data) that is usually treated differently in object-oriented data models. Therefore, types are also objects and their properties are modeled by relationships. This makes it equally easy for database users to model, view, access, and query the structural information and the data content of databases. It also simplifys the user-database communications by unifying the data-definition and data-manipulation languages into a single database language. A single database language permits the same database language constructs to be used in querying and updating both the schema and the data content of the database. The 3DIS data modeling formalism is a base on top of which the structures and the operations specific to an application environment (schema) can be defined. This however, does not have to be done in one step. Instead a hierarchy of concepts can be developed where the concepts closer to the ROOT of the 3DIS-kernel define more general structures common to many application environments. Concepts further away from the ROOT will support increasingly narrow application domains. The description below is essentially limited to the 3DIS-kernel (the ROOT and the first branches of this hierarchy), and introduces several notions that are common to all environments in which data models have to be defined. The 3DIS-kernel is the base on top of which 3DIS-schemas specific to applications will be defined. The 3DIS information modeling formalism is extensible in the sense that Abstract Data Types (ADTs) specific to applications can be easily defined and supported using the 3DIS modeling constructs. Previously, in applying the 3DIS information model to certain engineering and design applications Several domain specific ADTs are developed in terms of 3DIS modeling constructs. Among these, the definition of ordered sets, multi-lists, binary trees, and the hierarchic definition of design components in terms of 3DIS modeling constructs can be mentioned, details of which are reported in. Afsa90a Afsa89a Following are several characteristics describing the 3DIS information modeling formalism, some of which are distinct and not common to other object-oriented data models: Abstract objects can be directly represented independent of their symbolic surrogates. Objects at various levels of abstraction and of varied modality (different media) can be accommodated. Primitives are provided to support object classification and structuring. Basic relationships among objects are defined through the three fundamental abstraction primitives of classification/instantiation, aggregation/decomposition, and generalization/specification. Objects can be active as well as passive, in the sense that they can exhibit behavior. The approach taken to model object behavior is the abstract data type method. The important point is that procedures to manipulate data are represented as objects themselves. All identifiable information, of various kinds and levels of abstraction is viewed and modeled by the same constructs as objects. Therefore, descriptive information about objects (meta-data) is conceptually represented in the same way as the objects themselves. A consequence of object uniformity in 3DIS is the development of a single language with a small set of operations that can be used to query, access, and update both the schema and the data-content. The 3DIS is extensible to accommodate other kinds of abstractions; in particular it has been extended to support the definition of recursively defined entities and concepts, such as sets, lists, and binary trees. In the remaining of this section we first briefly describe the 3DIS information modeling formalism as it is originally defined, and then we specify specific extensions to the 3DIS model to support the Archon application domain. Our extensions include supporting entities and concepts (ADTs) required or beneficial to Archon and the typical applications for which Archon is intended. Objects and mappings are the two basic modeling constructs in the 3DIS information model, where mappings are a subset of objects. Relationships among objects are modeled by "(domain-object, mapping-object, range-object)" triples. Basic associations among objects are established through a set of predefined abstraction primitives. These primitives provide the basic organization and interrelation tools appropriate for application environments. Modeling capabilities of 3DIS can also be extended by a specific set of abstractions to represent certain concepts and entities of application environments. Every identifiable piece of information in an application environment corresponds to an in its 3DIS model. Simple, compound, and behavioral entities in an application environment, attributes of objects and relationships among objects, as well as object groupings and classifications are all modeled as objects. What distinguishes different kinds of objects in 3DIS model is the set of structural and non-structural (data) relationships defined on them.

The 3DIS model deals with objects, of all kinds, directly. However, to represent objects, we need designators (i.e. object-identifiers) by which we can refer to them. Each object has a globally unique that is generated by the system. Therefore, an object-id uniquely identifies an object within the entire model. Objects may be referred to via their unique object-ids, or via their relationships with other objects. An example where an object is referred to via its relationships with other objects is when we refer to as

The 3DIS model supports the three kinds of atomic, composite, and type objects. Notice that this separation of object kinds is important only at object definition time and not for object manipulation, retrieval, and evolution. Object-ids of short atomic objects are the (information contents of) atomic objects themselves. For long atomic objects, composite objects, and type objects, object-ids are chosen from a different domain than the one for atomic objects.

objects represent the symbolic constants. Atomic objects cannot be decomposed into other objects. The contents of atomic objects are uninterpreted by 3DIS, in the sense that they are either displayable or executable, without any further interpretation. Strings of characters, numbers, booleans, text, messages, audio, and video objects, as well as behavioral (procedural) objects are examples of atomic objects.

objects describe (non-atomic) entities and concepts of application environments. The information content of these objects can be interpreted meaningfully by the 3DIS system through their decomposition into other objects.

objects are a special kind of composite objects. Mappings can in general model arbitrary relations among two objects. They model both the descriptive characteristics of an object, e.g. a person's name via and the associations defined among objects, e.g. the association between a person and his manager via **Has-Manager** . They also model both single and multi-valued relationships, where a multi-valued mapping is defined from a domain element to a set of elements in its range. Every mapping is defined in terms of, and may be decomposed into: a domain type object a range type object an inverse mapping object The minimum number of the values it may return The maximum number of the values it may return objects specify the descriptive and classification information in an application environment; a type object is a structural specification of a group of atomic or composite objects. It denotes a collection of objects, called its together with the shared common information about these members. A type object is defined in terms of its members a set of mappings common to its members, called the subtype/supertype relationships between this type object and other type objects a set of operations defined on its members Therefore, type objects encapsulate the stored information about object sets. A type object can be a of other type objects (supertypes). Every member of a subtype is always a member of its supertype, but not vice versa. That is, members of a subtype are always a subset of the members of its supertype. Subtypes are defined by the enumeration of arbitrary members of their supertypes and inherit some of their supertypes' definition, namely the common mappings and operations. Member-mappings of a subtype may not have the same object-ids as member-mappings of its supertype(s). A type object can be the subtype of more than one other type object. Therefore, the subtype/supertype relationships among type objects can be represented by a directed acyclic graph (DAG). The fundamental associations among data and meta-data are captured through a set of *basic relationships* defined on objects. These relationships are established via the following built-in, basic mappings (and their inverses). Basic mappings support the definition of the three abstraction primitives of classification/instantiation, aggregation/decomposition, and generalization/specialization. is represented by *member/type* mappings that each relates an atomic or a composite object, e.g. Emp1, to its generic type object(s), e.g. PERSON, EMPLOYEE, etc. Therefore, (EMPLOYEE, Has-member, EMP1). is represented by *member-mapping/type* mappings that each relates a type object, e.g. EMPLOYEE, to a mapping of its members, e.g. Has-Name, Has-Spouse, Has-Address, etc. Therefore, (EMPLOYEE, Has-member-mapping, Has-Address). is represented by *subtype/supertype* mappings that each relates a type object, e.g. EMPLOYEE, to a more general type object, e.g. PERSON. Therefore, (PERSON, Has-subtype, EMPLOYEE). Basic mappings also support the definition of operations common to the members of a type (called procedures), namely their *constraint-evaluators* , *storage-transactions* , and *retrieval-transactions* . The 3DIS provides a rich extensible framework of meta-concepts to structurally organize the information in a complex environment. It introduces a specific generalization (subtype/supertype) hierarchy that contains a small number of predefined (structural) objects to organize the common structural information of application environments. is a predefined type object that is the root-node of this subtype/supertype hierarchy. Several other predefined type objects are connected to Root via the generalization hierarchy. (ROOT, Has-subtype, {TYPES, MAPS, OPERATIONS})

(TYPES, Has-subtypes, {STRINGS, NUMBERS, BOOLEANS})

(MAPS, Has-subtypes, {META-MAPPINGS})

(OPERATIONS, Has-subtypes, {CONSTRAINT-EVALUATORS,
STORAGE-TRANSACTIONS, RETRIEVAL-TRANSACTIONS}) This hierarchy is a unifying super-structure that fits on top and connects to the structural graphs of 3DIS data models. The resulting hierarchies support investigation of the structural information of application environment without requiring any preknowledge about them. It should be emphasized that the predefined structural objects do not introduce any specific data model-dependent description or classifications, but rather describe the generic kinds of structural information that is common to many specialized application environments. The resulting hierarchies are extensible. Users may add to or modify these hierarchies using the basic facilities of the 3DIS. Additional abstraction mechanisms needed by application environments can then be modeled by a generalization hierarchy and included in the 3DIS structural framework. One specific example is the addition of a recursion abstraction to support the definition of sets, lists, trees, etc. Afsa90a Afsa89a So far, the common modeling constructs mentioned as part of the 3DIS formalism cover the elementary relations, such as aggregation, classification and

generalization. To better support the representation of application information however, higher level information structuring concepts are required to enable easy definition of information models, and the operations that can be used to manipulate such information. In other words, it is quite desireable to be able to define and support certain ADT (structures and operations defined on them) using the modeling constructs of the information modeling formalism. This would have as a major advantage that information structures that reoccur in many systems, can be made to look identical by the Archon layer. Therefore, for example, the operations on a graph that represents a road map, are the same as the operations on a graph in a different system that represents an electricity network. As mentioned before, the 3DIS information modeling formalism is extensible in the sense that ADTs needed by Archon applications can be defined and supported by the 3DIS-kernel. Following are some modeling issues that are commonly required within a large range of industrial automation applications and some specific examples to show such needs. Engineering applications manipulate data that is usually modeled by *standard mathematical entities* . These mathematical entities and operations defined on them should be made explicit and supported. Since systems are designed and created by people that share a similar background in education, all these systems will reflect the predominant engineering practices of their field. These engineering practices are again based on a shared universe of mathematical notions, which imposes a relation between the data that is represented in the data model and the mathematical notions which will be used as a way to view such data. For example, a map can be viewed as a graph, and a control system may be viewed as a linear dynamic system, by a typical user, educated in the same engineering practices. The relation between data and the mathematical notion used to view it should be reflected and visible at the data model level, and operations on such mathematical notions must be supported. For example, if a map is viewed as a graph, the data model should support normal graph operations such as the shortest path. Similarly, if the user models a control system as a linear dynamic system the data model should be capable to link the data items to the notions of the theory of linear dynamic systems (e.g. state vector, system matrix, measurement matrix). At present our work concerns the definition of the following structures, and a set of operations to support them, within the 3DIS formalism. Graphs, vectors and matrices. Subtypes of graphs: DAGs, state graphs related to partial ordered sets. concurrency models (concurrency nets, Petri nets), defined in terms of graphs. The definition of notions as ‘plans’, ‘schedules’, etc. is expected to be done in terms of the models mentioned above. The operations on these entities will also be supported in the context of the type and map derivation operators (described in Section 5), used to establish a conversion between import and export schema.

The 3DIS/ISL is an object-oriented query and update language defined on top of the 3DIS model. This language provides a set of primitives to uniformly define, access, retrieve, modify, and activate (for procedural objects) the structural (meta-data) and non-structural (data) content of the 3DIS information-bases. The 3DIS/ISL includes a small set of simple but functionally powerful object-oriented primitives, called the "Object Specification Operations". These operations support the data definition and data manipulation of the 3DIS modeling constructs. In specific, the operations allow users to add new objects (that may be of kind atomic, composite, or type), to remove existing objects from a database, to create and destroy relationships among existing database objects, to retrieve objects and relationships among objects, to invoke behavioral objects, and to display objects on appropriate devices. The object specification operations contain "basic operations" and "extended operations". The basic operations are primitives to query and update the 3DIS information-bases, while the extended operations are somewhat compound operations (defined on top of the basic operations) to assist the programmar (user) of 3DIS/ISL language. For instance, set manipulation operations and the iteration of a command on the elements of a set are supported as extended operations. The set of object specification operations supported by 3DIS/ISL language is "complete" in the sense that it provides the expressive power to state any and all possible queries on objects stored in the 3DIS information base. Afsa89a The notion of Completeness introduced here is in analogy to the definition of "relational completeness" for languages defined for relational database theory. Voss91 Date81 Later to better support ISA, we will extend the 3DIS/ISL language. An example extension is the addition of a command to support the recursive search of related objects in a 3DIS-schema to answer a query on the existance of a relaionship between two 3DIS objects. Such a primitive command is also needed to be defined to support the recursive search of connected nodes in a graph to answer a query on the existance of a path between two nodes. Additional extensions to 3DIS/ISL are forseen to support the 3DIS/DIA machanism of information access and information sharing. In this Section we present the basic and extended operations. These operations are followed by their written description and some examples of their use. Due to the lack of time, the written description and examples provided are somewhat brief. Finally, a formal BNF-like definition of the 3DIS/ISL language is presented here. The 3DIS/ISL language defined here will serve as the base to later develop the 3DIS/DIA mechanism and "user-interfaces" to support the (human) user interaction to the Archon system. There are two main kinds of operations "basic operations" and "extended operations" defined. Here, in order to define the operations (commands) supported by the 3DIS/ISL language, first they are divided into smaller groups of operations (when their definition is similar), and then for each group a brief description and a few examples are provided. Also to simplify the desccriptions , only "simple primitive operations" are defined. This means that: first, the primitives are defined in terms of object-ids (rather than object-refs), second, they are defined on simple triples (rather than compound triples) and third, the triple of RETRIEVE command is atomic (rather than conditioned on the return values). However, if a primitive operation that is defined on triples is applied to a compound triple, the operation first changes the compound triple into a set of simple triples and then iterates on them. Once again notice that the separation in object kinds is only important in defining (by DEFINE command) and adding (by CREATE command) the objects to the information-base and not for their manipulation and retrieval. Following is the definition and some examples of the usage of basic operaions Examples provided for each operation assume that previous examples are executed.

The CREATE operation generates a new composite or type object, adds it to the database, and returns the object-id.

```
CREATE: object-id
CREATE(T, <short-name>): object-id          /* for types */
CREATE(M, <short-name>): object-id          /* for mappings */
CREATE(O, <short-name>): object-id          /* for members of types */
```

The object-id for a composite or type object is a globally unique identifier for that object. If the short-name suggested by user in the create command is globally unique then the system will recognize it as that object-id, otherwise a system generated object-id will be assigned to that object and returned to user. The system generated object-id is selected from a different domain than the typical domain of names used by programmer. In the following examples PERSON, STUDENT, FACULTY, Mary, Has-name, Has-phone#, Has address, and Has-advisor are being created.

```
CREATE (T, PERSON)
CREATE (T, STUDENT)
CREATE (T, FACULTY)
CREATE (O, Mary)
CREATE (M, Has-name)
CREATE (M, Has-phone#)
CREATE (M, Has-address)
CREATE (M, Has-advisor)
```

The above operations create eight new objects and assign their object-ids. The relationships that will be later defined on these objects (through the RELATE operation) determine the actual semantics of each object. For instance, the fact that STUDENT is a subtype of PERSON, Mary is a member of type STUDENT, and Has-name

is a mapping object for type PERSON will all be determined when these objects are further defined in terms of their relationships with other database objects.

The DEFINE operation generates a new atomic object and adds it to the database.

Define of short atomic objects:

```
DEFINE(S, "<shortstring>"): object-id     /* for short strings */
DEFINE(R, '<r>'): object-id                /* for reals */
DEFINE(I, '<i>'): object-id                /* for integers */
DEFINE(B, '<TF>'): object-id               /* for booleans */
```

Define of long atomic objects:

```
DEFINE(L, <short-name>, fn: filename): object-id    /* for long strings and texts */
DEFINE(A, <short-name>, fn: filename): object-id    /* for audio objects */
DEFINE(V, <short-name>, fn: filename): object-id    /* for video objects */
DEFINE(P, <short-name>, fn: filename): object-id    /* for procedures */
```

The object-id for a short atomic object is the object itself. For example, the object-id of an integer object is the integer itself. For long atomic objects, the short-name provided by programmar will be the object-id if it is globally unique, otherwise system will generate the object-id. The filename specified for long atomic objects is the file that contains the atomic object itself. Furthermore, since object-ids are globally unique identifiers, atomic objects cannot be defined redundantly and if so the system informs users that the object already exists in the database. For example, the number '1324' or the character string "Mary Smith", appears at most once in a 3DIS database.

```
DEFINE (P, admit, file22)
DEFINE (I, '1324')
DEFINE (S, "1080 Marine Ave.")
```

The above operations create three new atomic objects and return their object-ids.

The RELATE operation generates a relationship among objects and adds it to the database, where d is the domain-element, m is the mapping-element, and r is the range-element in the relationship. The arguments d, m, and r might have been CREATEd, or DEFINEd before this operation. Otherwise, any primitive operation that returns an object-id (or a set of object-ids in case of PICK operations) can replace any or all of these three arguments. The following examples define some relationships among database objects:

```
RELATE(PERSON, Has-member-mapping, {Has-name, Has-phone#})
RELATE(admit, Is-a-member-of, STORAGE-TRANSACTIONS)
RELATE(STUDENT,Has-supertype,PERSON)
RELATE(STUDENT, Has-member-mapping, Has-address)
RELATE(STUDENT, Has-storage-transaction, admit)
RELATE(Has-name, Is-a-member-of, MAPS)
RELATE(Has-name, Has-domain-type, PERSON)
RELATE(Has-name, Has-range-type, NAME)
RELATE(Has-advisor, Is-a-member-of, MAPS)
RELATE(Has-advisor, Has-domain-type, FACULTY)
RELATE(Has-advisor, Has-range-type, STUDENT)
RELATE(Has-advisor, Has-maximum-values, DEFINE(I, '1'))
RELATE(Has-advisor, Has-minimum-values, DEFINE(I, '0'))
RELATE(Mary, Is-a-member-of, STUDENT)
RELATE(Mary, Has-name, DEFINE(S, "Mary Smith"))
RELATE(Mary, Has-address, "1080 Marine Ave.")
```

Note that the RELATE operation is used to define relationships among all kinds of objects. The first RELATE operation in the above example is applied to a compound triple. As described above, this triple is first changed into a set of simple triples and then RELATE iterates over them. The result will be:

```
RELATE(PERSON, Has-member-mapping, Has-name)
RELATE(PERSON, Has-member-mapping, Has-phone#)
```

Also, notice that the atomic object "Mary Smith" is generated inside a RELATE operation. Every mapping specified in a RELATE operation must have been defined in the database previously in terms of its domain-type and range-type, so that the domain and range elements of the specified relationship can be verified. Inverse

mappings and inverse relationships are always automatically defined by the system.

The UNRELATE operation destroys the specified relationship from the database. If the specified relationship in an UNRELATE operation does not exist in the database, then this operation has no effect. The operation UNRELATE(Mary, Has-address, "1080 Marine Ave.") simply destroys that relationship and leaves the objects and intact. Notice that all mappings (e.g., Has-address) are assumed to be multi-valued unless the minimum and maximum number of values they may return are explicitly restricted in their definition. If the minimum and maximum number of values for Has-address are specified as 0 and 1 respectively, then in order to insert a new address for Mary, user must first UNRELATE the previous address and then RELATE the new one.

The DELETE operation simply removes the specified object from the database. If the object that is being deleted participates in relationships with other objects then those relationships will also cease to exist, i.e., they will be UNRELATEd. For example, DELETE(Mary) removes the object Mary from the database, and UNRELATEs the following relationships (and their inverses):

```
UNRELATE(Mary, Is-a-member-of, STUDENT)
UNRELATE(Mary, Has-name, "Mary Smith")
UNRELATE(Mary, Has-address, "1080 Marine Ave.")
```

However, the objects referred to by the composite object STUDENT and the atomic objects "Mary Smith" and "1080 Marine Ave." remain in the database, and even the fact that Mary was the only object RELATEd to them does not affect their existence.

The DISPLAY operation, displays objects on the specified device. As mentioned earlier, all atomic objects are displayable. For example, the information contents of strings of characters, numbers, digitized audio or video objects, behavioral objects, etc. may all be displayed on appropriate devices. DISPLAY(admit, printer) prints out the procedure admit. Composite and type objects are not displayable except in terms of the atomic objects RELATEd to them.

The RETRIEVE operation queries the database by investigating the relationships defined among database objects. Any combination of the d, m, and r elements in a RETRIEVE operation may be replaced by a question mark "?". For instance, RETRIEVE(?, m1, r1) where m1 and r1 elements are object-ids, returns the set of all simple triples whose mapping and range elements are m1 and r1. For example, RETRIEVE(Mary, Has-name, ?) returns {(Mary, Has-name, "Mary Smith")}, and RETRIEVE(Mary, ?, ?) returns {(Mary, Has-name, "Mary Smith"), (Mary, Has-address, "1080 Marine Ave.")}. For the exceptional case where all three elements are "?", RETRIEVE returns the set of all existing relationships in the database. If no element is replaced by "?", then if the specified relationship exists in the database, the relationship itself is returned as answer, otherwise an empty set is returned.

The PICK-D operation projects simple triples on their D (domain) elements, and returns the corresponding object-ids. For example, if the following relationships are defined in a database:

```
RELATE(John, Has-phone#, DEFINE(I, '743-1234')),
RELATE(Mary, Has-phone#, '743-1234'),
```

then:

```
PICK-D(RETRIEVE(?, Has-phone#, '743-1234'))
```

returns {John, Mary}.

The PICK-M operation is similar to PICK-D except that it projects simple triples on their M (mapping) elements.

The PICK-R operation is similar to PICK-D except that it projects simple triples on their R (range) elements.

The EXECUTE operation invokes the specified behavioral object. The system issues an error message if the specified object is not executable. Parameters may be passed to behavioral objects upon invocation. The following examples show the content of two files GSTU1 and STAD1, respectively:

```
procedure STATUS-IS-GRADUATE
    var   student : object-identifiers
    begin
      for every student in
        PICK-D(RETRIEVE(?, is-a-member-of, STUDENT)) do
          if student is in
```

```
        PICK-D(RETRIEVE(?, Has-status, DEFINE(graduate)))
      then
        print(student, "is a graduate-student.")
    end          end


procedure SELECT-ADVISOR (student, advisor: object-id)
  var   number : integer
  begin
    for every number in
      PICK-R(RETRIEVE(Has-advisor, Has-max-value, ?)) do
        if count(PICK-R(RETRIEVE(student, Has-advisor, ?)))
          >= number
        then print ("Enough advisors are already defined for", student,".")
        else RELATE(student, Has-advisor, advisor)
      end
  end
```

Following are the commands to define these procedures as RETRIEVAL-TRANSACTION and STORAGE-TRANSACTION objects on the type STUDENT.

```
DEFINE (P, g-student, GSTU1)
RELATE (STUDENT, Has-retrieval-transaction, g-student)
DEFINE (P, s-advisor, STAD1)
RELATE (STUDENT, Has-storage-transaction, s-advisor)
```

Now, EXECUTE(g-students) invokes the procedure STATUS-IS-GRADUATE. This procedure prints out the names of all graduate students. EXECUTE(s-advisor, Mary, Susan) invokes the procedure SELECT-ADVISOR passing the two parameters of Mary and Susan. This procedure first checks the maximum number of values that can be returned by the Has-advisor mapping and compares it with the number of advisors already assigned for Mary. If the number exceeds the maximum, then the program issues an error message stating that Mary already has enough advisor(s). Otherwise, it defines Mary's advisor to be Susan. Notice that the maximum number of values for the Has-advisor mapping was previously defined to be "1", so, Mary cannot have more than one advisor. Since behavioral objects are treated as atomic objects, users are responsible for passing the correct number and type of parameters to procedures.

The ADDNAME operation defines new object-names for existing objects.

```
ADDNAME (John, /Johnny/)
```

The new user defined object-name /Johnny/ is also returned to user to indicate its acceptance by the systema as a globally unique identifier. Now, John and /Johnny/ refer to the same object and can be used equally (in place of each other) in future commands.

The DROPNAME command deletes object-names earlier added (using ADDNAME command) from the database.

```
DROPNAME (John, /Johnny/)
```

Most database primitives can be directly formulated with the "basic operations" in specific using the RETRIEVE, PICK-D, PICK-M, and PICK-R commands. These operations umplicitly support the use of "existential quantifier" as described in the relational calculus languages. However, formulation of some complicated database queries may require additional operations such as the "universal quantifier". The FOR-ALL operation defined as an extended operation plays the role of the universal quantifier.

```
FOR-ALL <variable> IN set-of-object-ids DO <command>: A-SET
```

The FOR-ALL operation performs the following task: (1) repeats the execution of the <command> for n>=0 times, where n is the number of object-ids in the <set-of-object-ids>, instantiating the <variable> to a different member of <set-of-object-ids> each time. (2) returns A-SET whose members are the result of the application of the <command> to the elements of the <set-of-object-ids> as
described in step (1).

As a result, depending on the <command>, the returned set (A-SET) is either: a set-of-object-ids, or a set-of-set-of-object-ids, or a set-of-simple-triples. To assist the programmar, set-manipulation operations of UNION, INTERSECTION, and DIFFERNCE are also supported as extended operations. These operations (with obvious

description) are usually used together with the other basic retrieval operations.

UNION(set-of-object-ids, set-of-object-ids): set-of-object-ids
INTERSECTION(set-of-object-ids, set-of-object-ids): set-of-object-ids
DIFFERENCE(set-of-object-ids, set-of-object-ids): set-of-object-ids

Two more set-manipulation operations are also supported:

UNION*(set-of-set-of-object-ids): set-of-object-ids
INTERSECTION*(set-of-set-of-object-ids): set-of-object-ids

UNION* and INTERSECTION* operations are defined on a set of sets and return a set which is the result of UNION or INTERSECTION of elements of the given set, respectively. A last operation, MERGE supports merging the pieces of results into the format requested by users.

MERGE(sst: set-of-set-of-simple-triples): set-of-simple-triples

The MERGE operation is defined on a sequence of sets and returns a set that is the cartesian product Date81 of those sets.

The syntax of the 3DIS/ISL language is specified as a variation of the BNF formalism, as a series of productions. Each production has a left hand side, which are seperated by the symbol "::=". The left hand side is the name of a non-terminal symbol being specified. The right hand side is a series of clauses, seperated by the symbol " | ", and representing alternative expressions for the left hand non-terminal. Non-terminals on right hand side are enclosed in "< >" pairs. Enclosing items within "[[ ]]" pairs indicates that the enclosed items may appear zero or one time. Descriptions and comments are enclosed within "{* *}" pairs.

The 3DIS/DIA (3DIS/Distributed Information Architecture) is a model and mechanism developed to support the distributed information sharing among loosely-coupled semi-autonomous cooperating agents within Archon. 3DIS/DIA uses the 3DIS information modeling formalism and the 3DIS/ISL language as the base. Design of the 3DIS/DIA is rooted in the concepts and mechanisms introduced in the Federated information architecture work. Heim85 IEEE87 The 3DIS/DIA formalism is still at the design and modeling stage. Specifically, the design of the Archon community dictionary and the type and map derivation operators are incomplete. We expect our study and experimentation with the Archon application domain to help us determine what is required to be supported by these components of the 3DIS/DIA formalism. The Federated architecture was primarily introduced to support cooperation among local databases in an office environment. Federated Architecture of Information Modeling and Management is an approach to the coordinated sharing and interchange of computerized information among autonomous information systems. Information systems (also called agents), while independent, are united into a loosely coupled federation in order to share and exchange information. This architecture provides mechanisms for sharing data and operations defined on data, and for combining information from several information systems and coordinating activities among autonomous information systems. For each information system in the Federated architecture a "private schema", an "export schema", and an "import schema" is to be developed. The private schema represents the structure of the private information stored in the information system. The export schema represents the structure of the onformation an agent wishes to share with other agents. The import schema represents the structure of the information an agent wishes to receive, which is exported by other agents. Any exchange of information is done after the establishment of a negotiation between two agents. This negotiation involves determining which information to be exchanged and which rights are associated to the receiver agent with regards to which part of the information (rights to read, rights to write, rights to transfer these rights to other agents, etc.) . Using a predefined negotiation protocol, agents can contact each other directly to negotiate the export and import of information on types and maps. Once the negotiation on a type (or map) is completed a contract will be established, further access to the information content of this type (or map) is direct and without the overhead of negotiation. It is important to notice that the actual transfer of data occurs when the importer (receiver) tries to access the instances of the type (or map) which is dealt with the same way as accessing the content of the type locally within an agent. The negotiation contract between two agents guarantees that the exporter will not modify the definition (structure or semantic) of the type (or map) unless it informs the importer. Later modifications to the contract, initiated by exporter or importer, may also be performed through the negotiation protocol. The federation framework has a single federal dictionary. This dictionary maintains a list of information systems involved and the information on federation among them. The federal dictionary is represented as a separate agent (information system in the federation). Its administrative tasks can also be distributed over the agents participating within the federation. The federal dictionary contains the addresses of the agents participating in it. It also contains the negotiation protocols and the contracts that have been negotiated between agents. The federal dictionary is also used as the general store for other static information that concerns the entire federation. Import and export schemas of all agents involved in federation are stored in federal dictionary. To reduce dependency and support agent’s autonomy, this information can then be copied to an agents once it joins the federation. A set of type and map derivation operations are also introduced by federated architecture to be applied to derive (and define) an agent’s import schema based on other agents’ exprot schemas. The derivation operators are essentially set operators from the relational calculus domain. These were chosen by federated architecture because they make it possible to resolve a large class of structural information (description and classification information), while at the same time they are easy to realize. The principal type derivation operators introduced for the federated architecture are: concatenate (instances of two types), subtract (instances of a type from another), cross product (of instances of several types), and subtype (generated through the application of some predicate to an existing type). The principal map derivation operators are: composition (of two maps), inversion (of a map), extension (of a map from the type it is defined on to its supertype), restriction (of a map from the type it is defined on to its subtype), cross product (of the two sets of values resulted by two different maps as the result of a new map), and the discrimination, selection, and projection that are complex operations specific to the maps defined on the types involved in cross product of several types. The federated architecture as described above will then support the cooperation of agents. Loosely-coupled semi-autonomous agents in the federation will then have the following properties: No agent can be forced to provide information. (*negotiation enforced) Each agent determines which part of its data it wishes to share with other agents. (*export schema) Each agent determines what information it wishes to receive and which agent (and in what form) it wishes to receive it. (*import schema and type and map derivation operators) Each agent can build its own local schema of existing data and there is no global schema for the federation. (*private schema) Each agent has the freedom of association or leaving the federation. (*federal dictionary) Each agent has the freedom to change its policy and share a different part of its data at any time. (*export schema, federal dictionary, and negotiation) The main goal of the 3DIS/DIA is to provide a framework of interconnected agents in which they can maintain their autonomy, yet support mechanisms for a substantial degree of information sharing, and while minimizing the central authority. The 3DIS/DIA adopts an approach similar (but more limited) to Federated architecture to support the distributed information access and information exchange among semi-autonomous Archon agents. 3DIS/DIA is defined in terms of three schemas describing the structure of the information stored and shared by agents, a set of derivation operations used to describe these schemas, and a dictionary containing information on agents involved in distributed information sharing and the shared information itself. It uses the 3DIS information modeling formalism as a base to represent the "private", "export", and "import" schemas, and the 3DIS/ISL language primitives to develop and implement the type and

map derivation operators. Each agent is autonomous and is described by three schemas: a private schema, an export schema and an import schema. Each schema is described by a generalization hierarchy (that contains types, subtype/supertype relationships among types, and the mappings and behaviors (operations) defined on each type). Both export schema and import schema can describe application specific information ('IS information representation'), which is accessed by the underlying IS. The private schema of an agent represents a complete description of the information contained in the agent. This schema also contains a small collection of information and transactions relevant to the agent's participation in the archon community, including: the identifier name of the agent and its network address, the primitive operations for manipulation of types and maps in each schema, and agent's import and export schema. The information about agents' participation in the archon community is also stored in the archon community dictionary. The export schema of an agent represents the description of the information that the agent is willing to share with other agents in the archon community. Other agents can import the export schema of an agent and use it as a guideline to the information that they may access. If an agent wishes to limit the access to certain portions of its export schema only to certain specific other agents, the agent can represent this restriction by placing access control on the relevant types and maps of its export schema. Access control lists contain the agents' identifier names who can use the information in the export schema and the type of access 'read' or 'read/write'. The import schema of an agent contains the information that the agent desires to use from other agents. Each imported element (type or map) has a derived definition property, specifying how the imported element is derived from the underlying exported element(s). To share information agents can perform 'schema-importation'. Any importation of information must then be explicitly negotiated with other agents. The information about the export schemas of other agents in the archon community can be gained from the archon community dictionary. The archon community dictionary, (which can either be a separate element or distributed among the available agents), contains the information about all agents involved in the community that an agent need to share information with. The information stored in the dictionary consists of at least a list of the agents in the archon community, their names and network addresses. Import and export schemas of all agents invonved in the Archon community are also stored in the dictionary. If a data sharing negotiation protocol to be supported by 3DIS/DIA then, the protocol itself and the negotiated contracts will also be a part of the archon community dictionary. Once an agent is allowed (has a contract with another agent) to import some types and maps (mappings defined on types), the agent is allowed to restructure that information to suit its purposes. To support this the federation architecture for information exchange provides a set of "derivation" operators to manipulate and modify the type and map definitions. We forsee the need to extend the type and map derivation operators introduced in the federated architecture in order to support the restructuring (convergence) necessary for the exchange of information among Archon applications.

The implementation of ISA involves the development of ISA's components. These components are: (1) An information modeling formalism in which an agent can describe the information it wishes to share with the community, (2) A query language with the expressive power of relational calculus to access and manipulate the available information, and (3) a methodology by which agents can know about the information stored by other agents and which can be accessed remotely. Sections 3, 4, and 5 of this technical report respectively described the design and presented a specification of these components. This section first addresses some important issues concerning the implementation architecture of ISA and then describes the approach taken by UvA for this development. The first step in the design of an implementation architecture for information sharing is to decide on: (1) where the information that an agent wishes to share is to be stored and (2) which element of the Archon architecture is responsible for the execution of queries and the interpretation (parsing) of a query expression. These two questions have to be first addressed in relation to the level of distribution of information and query processing within the entire Archon system. From this point of view the two main questions can be reformulated as: Is the information to be shared stored locally (in the agent that has generated it) or distributed (by broadcasting it), and can other agents store local copies of the received information (for efficiency reasons). Is the interpretation of the query done locally, remote, or maybe in some distributed fashion. Formulated in this way, we have raised the main questions involving distributed information systems. In fact, within the scope of the Archon project these questions have to be also addressed in the wider context of the cooperation paradigm, not simply considering sharing the static information model, but for example, also considering the execution of tasks. For our purposes, within the limited context of information sharing (which means not considering the dynamic aspect of generating information), we can address these questions using the (simple) solutions developed for distributed information systems. Simple solutions will suffice for Archon applications since so far we have no indication that ISs may share enormous amount of information, and it is for that type of systems only that more complex answers should be provided. For the implementation of ISA similarly, the two questions raised above must be tackled in relation to the structure of a single "agent" which consists of an already existing Intelligent System and an Archon layer on top of it. Within this context the two main questions will be restated as follows: Is the information (to be shared) stored in the IS itself or in the Archon layer. Is the query executed at the IS or at the Archon layer. This generates four different possibilities. Out of these four, the one where the information to be shared is stored in the Archon layer and the query is interpreted at the IS can be dismissed for clear inefficiency reasons, leaving the other three options open. Each of these options are discussed below. In this situation we are using the Archon layer completely as an interface layer, which simply transforms information requests that are formulated in a common language at the Archon level (as defined by the ISA modeling constructs and query language), into information requests that are understood by the IS. However, there are serious disadvantages to this approach. The most fundamental one is the fact that the interpretation of a query is done by the IS itself. Therefore, only those information queries can be formulated that can be dealt with by one IS. Also, there are a number of practical difficulties. It is necessary to construct a complete mapping of the operations (query language) understood by each IS to the ISA query language. If the information to be shared is stored in the IS, and if the IS is not capable to interleave activities then the information request have to be sequenced in the same manner as requests for task execution. This means that no immediate response on information requests can be guaranteed. In this approach the information remains within the IS, but the queries are processed by the Archon layer. The archon layer must know of the storage model for the information inside the IS. In processing queries it will use low level operations to access the IS's information structures, and itself performs the operations required by the query. This approach has the fundamental disadvantage of the first approach, which was that a single query can only refer to entities within a single IS. There are however some benefits: ISs need not support query evaluation, so this approach can also be used for systems that have not been developed as expert systems, such as numeric simulation programs. It is no longer necessary to translate the query language of the IS to the ISA query language. Information to be shared remains locally in the IS, no extra copying is needed. The main disadvantages of this approach are: Since the underlying IS is unlikely to have been designed for efficient, general purpose information retrieval, the operations to access its data structures may not be very efficient. This will certainly lead to a serious performance degradation if many queries are formulated over the same data set, forcing the IS to transfer many times the same data to the Archon layer. No immediate access can be guaranteed, since the IS may be busy performing some basic tasks. In this approach the Archon layer performs the function of a distributed information system, or distributed blackboard. Information that an IS wishes to share with other agents will be copied to the Archon layer and gets updated when necessary. This approach has the following advantages: Immediate access can be guaranteed by the Archon layer, since everytime a new information request is received a new process can be created. Distribution of information, maintaining local copies, performing broadcasts, etc. is all under the control of the Archon layer, and can therefore be optimized by Archon. Simple implementation of a module at the Archon layer to support this approach is feasible. A standard structure can be designed for this module, interface problem is a matter of adding code to the IS to copy information to the Archon layer. The main disadvantage here is that all shared information has to be copied to the Archon Layer. This is a serious disadvantage if there is a lot of shared information that is regularly updated. All of the options outlined above are feasible, and each may in some situation be the best solution. Should there be a need, it is always possible to implement all three options as different alternatives if resources make it possible within the current project. For the implementation of ISA however, the last option described in section 6.3 above is the most attractive to choose. By adopting this option, the definition and manipulation of the information to be shared is brought to the Archon layer and a unified approach can be applied to the modeling, language, and distributed access of information problems. The

implementation of ISA is then realized through the development of AIM (Agent Information Management) module to be included in the Archon layer of each cooperating Archon agent. The development of ISA for Archon primarily involves the implementation of its three main components. Therefore, first the implementation of the extended 3DIS information management system,second the implementation of the extended 3DIS/ISL language, and third the implementation of the 3DIS/DIA must be accomplished. An experimental prototype implementation of the 3DIS information management formalism and the 3DIS/ISL language is already developed at the University of Amsterdam. All basic specification operations (described in section 4.1.1) are coded. A few high-level operations are also implemented (type-create, member-create, etc.). A description of basic and high-level operations and a description of an example "demo" database can be found in The 3DIS/ISL User Manual. Afsa91

**** FIGURE 1 ****

A major step in the implementation of ISA involves developing AIM modules for Archon ISs. The AIM module of each IS consists of three schemas (meta-data, structural definition) and a copy of only that part of the IS data that is to be shared with other Archon ISs. The three schemas are the private, export, and import schemas as described under the section 5.2.2 of this technical report. The AIM module can then perform update and retrieval queries issued either by the local IS or by a remote agent. Queries are stated in 3DIS language, extended with domain specific, but application dependent concepts. The distributed information access and sharing of ISA is then supported through export and import of schema among AIMs. We develop AIM as a module in agent's Archon layer (see Figure 1), rather than a data structure "layer" since unlike other modules in Archon Layer that support the communication among ISs, it does not seem necessary to have all messages to and from the IS pass through AIM. Also, implementing AIM as a separate Unix process has the advantage of not delaying the other communications when AIM is executing updates and retrievals which may be quite frequent. The only disadvantage here is the more overhead (but constant time) that it results. A first step to deal with the problem that two agents have different models of the same thing is to develop an appropriate formalism in which the information can be described that an agent has or needs. This is done by means of the modeling constructs described in the paragraph on 3DIS. The ISA applied to the problem is as follows. First we have to identify the private, import and export schema of each agent. The private schema of each agent is of no concern to us here, so we will not consider them. For the CAD agent the export schema describes the boundary representation. This specifies the adjacency relations between faces, vertices and edges. For the vision agent the export schema describes the edge graph that has been created on the basis of the observed image. In this example neither the CAD agent nor the vision agent need any import schema. The HLI agent has one import schema, defining an edge adjacency graph. For this import schema it has two sets of type and map derivation operations, one for the import of the schema provided by the CAD agent, and another for the import of the schema provided by the vision agent. Figure 2 shows this, in the same format as Figure 1 and with some details described in section 5.2. Now that the HLI has imported the schema and has defined its own private version, it can access the data

available in the other systems simply by reference to its own import schema definition. The Archon layer on top of the HLI will execute the functions that are associated with the map and type derivation procedures so that the vision agent and the CAD agent simply get information requests in terms of their own export schema. The HLI will issue information requests to its Archon layer in terms of its import schema definition. The advantage of this approach is that the HLI is completely shielded from the actual information representation used in the other systems. This also implies that if a different system is introduced that produces similar information, this system can easily be integrated, just by using the map and type derivation operators to convert the schema used by this system to the schema used by the HLI. Not yet resolved, but under study, are the type and map derivation operators that are needed for this application. The type and map derivation operators described in 5.2.4 do not suffice. These operators deal essentially with the notion of set only. The operations that will be introduced for the high level modeling constructs described in 3.4 will be better suited, since then the notion of graph will have been made explicit at the Archon level.

**** FIGURE 2 ****

We have described an architecture for sharing and exchange of information among heterogeneous and autonomous agents in a loosley-coupled, multi-agent system. The architecture is tailored to support the cooperation of Archon agents. Each Archon agent consists of an Intelligent System (IS), that is pre-existing or built for the task, and the Archon Layer (AL) which provides the means for cooperation and communication. The control is decentralized and agents either perform actions on the real world or they provide other agents with information, for example the diagnostic system agent. We have described the ISA architecture in terms of its three main components, the 3DIS database model, the 3DIS/ISL query and update language, and the 3DIS/DIA distributed information architecture. We have given an overview of an implementation strategy and tried to illustrate our proposed architecture using an example from the

$LIST$