PARTIALLY SHARED VIEWS
A Scheme for Communicating among Groups that use Different Type Hierarchies

Jintae Lee
Thomas W. Malone

CCSTR #111 3401 SSWP
Supersedes 3291

To appear in the ACM Transactions on Information Systems, January, 1990.

Partially Shared Views: A Scheme for Communicating among Groups that Use Different Type Hierarchies

Jintae Lee and Thomas W. Malone
Massachusetts Institute of Technology

Many computer systems are based on various types of messages, forms, or other objects. When users of such systems need to communicate with people who use different object types, some kind of translation is necessary. In this paper, we explore the space of general solutions to this translation problem and propose a scheme that synthesizes these solutions. After first illustrating the problem in the Object Lens system, we identify two partly conflicting objectives that any translation scheme should satisfy: preservation of meaning and autonomous evolution of group languages. Then we partition the space of possible solutions to this problem in terms of the set theoretic relations between group languages and a common language. This leads to five primary solution classes and we illustrate and evaluate each one. Finally, we describe a composite scheme, called Partially Shared Views, that combines many of the best features of the other schemes. A key insight of the analysis is that partially shared type hierarchies allow "foreign" object types to be automatically translated into their nearest common "ancestor" types. The partial interoperability attained in this way makes possible flexible standards where people can benefit from whatever agreements they do have without having to agree on everything. Even though our examples deal primarily with extensions to the Object Lens system, the analysis also suggests how other kinds of systems, such as EDI applications, might exploit specialization hierarchies of object types to simplify the translation problem.

Computer-based systems often use various types of messages, forms, and other objects to communicate information. When all the people who wish to communicate with each other use exactly the same types of objects, translation problems do not arise. However, when people want to communicate with others who use different kinds of objects, some kind of translation is necessary. The needs for such translation seem likely to become increasingly common as more and more diverse kinds of systems are linked into heterogeneous networks.

We have been particularly concerned with one instance of this problem that arises in the context of template-based office communication systems [17, 24]. The problem is how users of such systems can communicate with other users who have a different set

of templates. Other examples of the problem arise when users of different databases exchange data or queries [2], or when different companies wish to use Electronic Data Interchange (EDI) standards to exchange business forms such as purchase orders and invoices [23]. In general, of course, the problem of automatically translating between independently defined object types is quite complex. It may involve, for example, making subtle inferences about the intentions and usage contexts of very different user communities.

In this paper, we will propose one way of taxonomizing the space of general solutions to this translation problem and describe a specific scheme that synthesizes the different types of solutions in this taxonomy. Our primary goal has been to design extensions to the Object Lens system [10] that allow different groups to communicate with each other when (1) the groups use some, but not all, of the same types of objects, and (2) the object types used by each group may change over time. Even though much of the scheme is straightforward, it suggest some useful and non-obvious extensions for systems like Object Lens and for other kinds of data interchange as well.

One of the most important general lessons of our analysis is that several novel and attractive translation schemes are possible when the object types are arranged in an inheritance hierarchy with certain types of objects being regarded as specializations of others. In particular, unknown object types can be automatically translated into the nearest "ancestor" types they share with the receiving system. One other interesting aspect of this approach is that it translates the contents of foreign objects into object types for which the receiving system already has actions defined. In the language of object-oriented programming, this means that partial interoperability is attained by translating object *fields* without needing to translate object *methods*.

In the first section of the paper, we illustrate the problem as it arises in the context of the Object Lens system. In the second section, we state the problem more precisely in terms of the objectives that we want its solution to satisfy. In Section 3, we explore the space of possible solutions by suggesting a dimension along which to partition the space and examining a representative solution for each class. In Section 4, we propose a new scheme that combines most of the desirable features of the solutions we explored and in Section 5, we describe our implementations of the scheme in the Object Lens

system. Finally, we hint at the implications that this research might have for other contexts such as Electronic Data Interchange.

1. Illustration: Inter-group communication in the Object Lens system

1.1 *Description of the Object Lens system.*

Object Lens is a general tool for cooperative work and information management. It provides flexible facilities for storing and manipulating representations of many different kinds of "objects," such as messages, people, projects, and tasks. For example, it lets people specify rules to automatically filter, sort, and prioritize electronic messages they receive; and it helps people find and summarize objects in a database. The system is described in much more detail elsewhere [10].

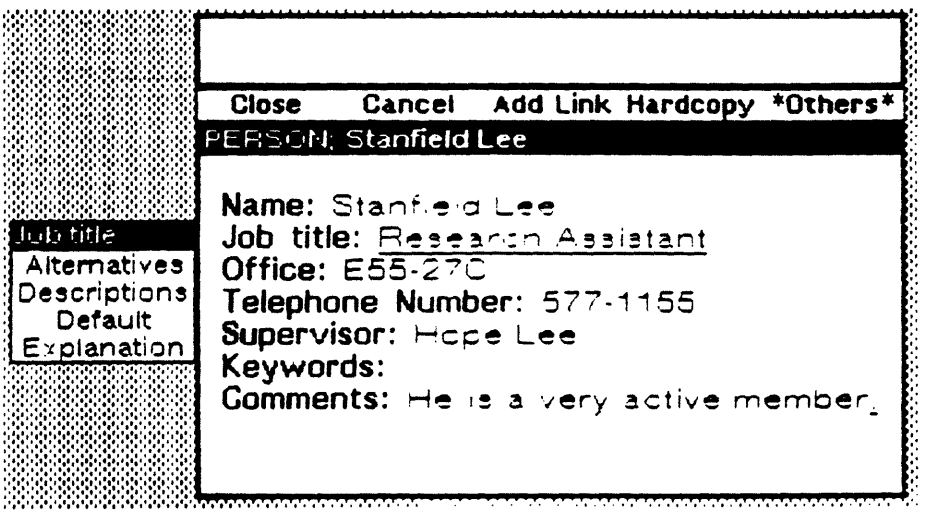


Fig. 1. A template for an object type, PERSON, is displayed in an object editor.

For our purposes here, a central feature of the system is that it depends on a set of semi-structured templates for different types of objects. For instance, a person object contains fields for name, job title, telephone number, and address (Figure 1), while a message object contains fields for To, From, Subject, and Date. The display-oriented object editor, shown in Figure 1, uses pop-up menus to suggest alternative values for

different fields, but users can put any value they desire in any field. A similar editor is used to create rules for automatically processing objects of a given type based on the contents of their fields. Users can also easily summarize the contents of collections of objects in various formats such as tables and trees. In addition to fields and values, a template has several other properties such as a display format and a set of actions that can be applied to it. For instance, some actions are appropriate for all objects (e.g., Show or Hardcopy) while other actions are appropriate only for certain types of objects (e.g., Answer or Forward for message objects).

The set of object types currently in use by our research group at MIT is shown in Figure 2. As shown in the figure, the templates for different object types are arranged in a network with some templates designated as "subtypes" (or specializations) of others. The subtypes of a given template automatically inherit from the parent template its field names, actions, and other properties. Any subtype may also, in turn, add new fields or override any of the property values inherited by the parent [4]. For example, the Action Request message type has the field Action Deadline in addition to the usual fields such as To, From, and Subject. Its child, Bug Fix Request, inherits these fields and has additional fields such as Bug Name and Severity.

We will refer to this network of object types as a *type hierarchy* even though it need not be a strict hierarchy. The *multiple inheritance* capabilities of the underlying knowledge representation system that we use [22] allow one message type to inherit properties from more than one parent.

1.2. Communicating with other groups.

In the simplest version of Lens, all users of Lens in a local group are expected to share the same network of object types (though individual users can customize for themselves the display properties of a shared object type). The problem arises when users of an Object Lens system want to share objects with others who do not have definitions for the object types to be shared. The simplest case of this problem arises with exchanging messages, and we will illustrate the problem in this context.

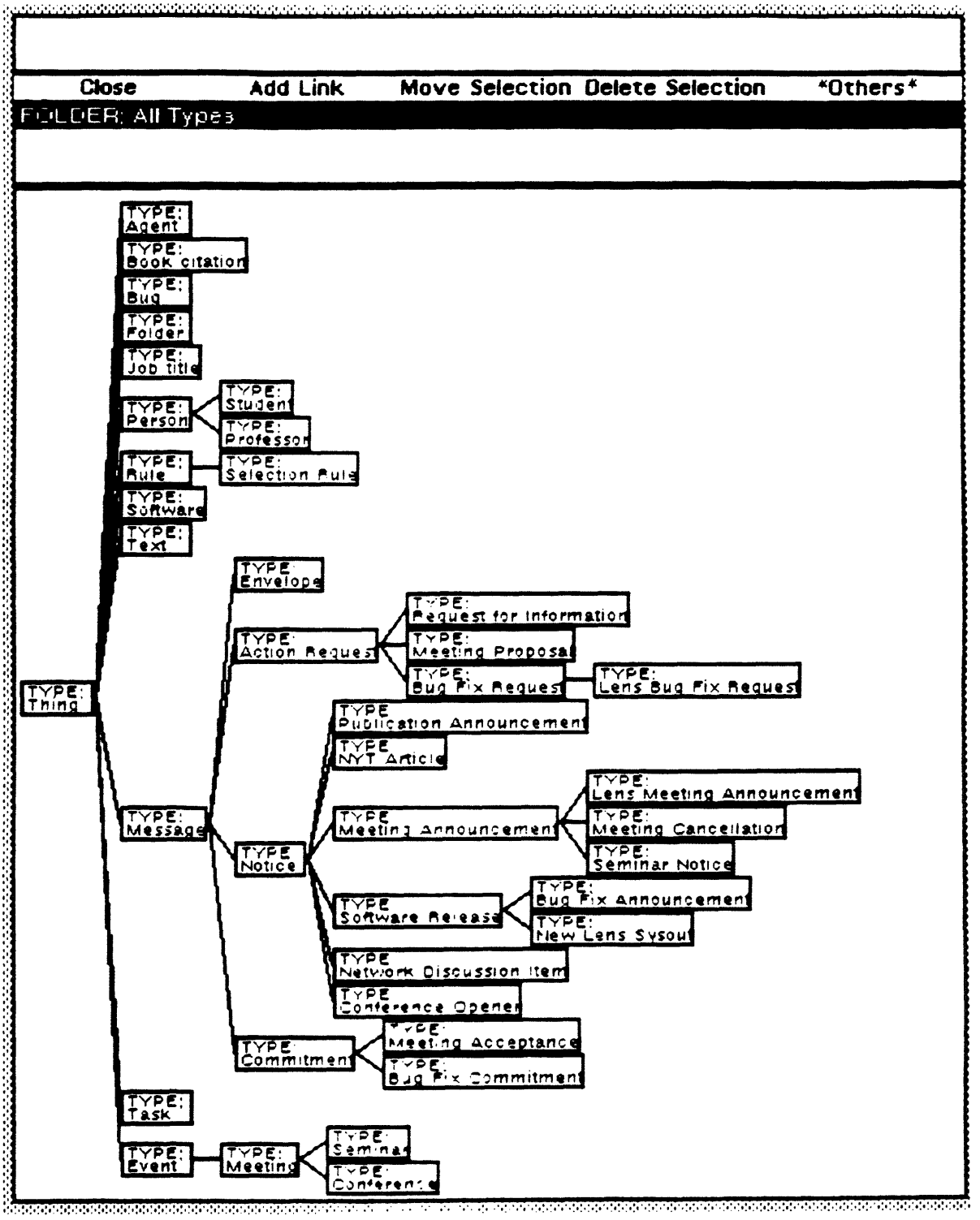


Fig. 2 The object type hierarchy used at MIT

Since Lens users are connected to national electronic mail networks, they often communicate with other people who do not use Lens at all, and they sometimes communicate with users of Lens at other sites that use a different set of message types.

One solution to this problem was the following: When a Lens message is sent--even within the local group--it is sent as an ordinary message with only the standard mail header fields in its header. All the other fields specific to this message type are sent as the first few lines of what the mail transport system regards as the body of the message. The body of the message also includes a field called "Message Type" and another one called "Text" that contains all the remainder of the message. Thus any electronic mail recipient can read a message from a Lens user. If the message is one of the specialized types with extra fields, these fields (including the field names followed by colons) will simply appear in the body of the message.

When a Lens user receives a message, the message is first checked to see whether it contains a field called Message Type and whether the name in that field is one recognized by the local group. If so, the message is treated as being of that specialized type. If not, the message is treated by Lens as the most general message type of all--Message. Thus Lens users can receive messages from any connected electronic mail user. These messages are not processed by any of the special purpose rules set up for particular kinds of messages (such as Meeting Announcements or Bug Fix Requests). They can, however, be processed using rules that check fields present in all messages (such as To, From, Subject, and Text). Thus the system degrades quite "gracefully," taking advantage of as much of the Lens functionality as possible, even when communicating with non-Lens users.

With this scheme, Lens users can also receive and process specialized message types from Lens users at other sites--or even from non-Lens users who simply send messages with the appropriate field names typed into the body of the messages. When the sending site and the receiving site have exactly the same definitions for a given message type, this communication works as desired. A similar scheme would, of course, work for other types of objects besides messages. Instead of translating unknown object types into "Messages", however, they should be translated into the most generic type of object (called "Thing" in Object Lens). Some of our examples in

the remainder of this paper will involve only messages; others will involve more general object types.

There are two kind of problems, however, that can occur with schemes like this:

(1) If the sender and the receiver use the same name for what are actually different object types, then the incoming objects may have either extra fields, missing fields, or both. In these cases, the extra fields are inserted in the text of the message (or in the "Comments" field of an object) and the missing fields are simply treated as being empty. Even if fields with the same names are present, the sender may be using these fields in ways that are different from what the receiver expects.

(2) If the sender and receiver use different names for what are in fact the same object types (e.g., Bug Report and Bug Fix Request), then humans can read each other's objects satisfactorily, but none of the automatic processing that could otherwise have been applied to the objects will be invoked.

No disasters occur in either of these cases, but the Lens system fails to be as helpful as it might be. The problem we will explore in the remainder of this paper is how to design systems like Lens that can retain as much functionality as possible in communication between groups without imposing excessive burdens on the different groups to coordinate their changing document type definitions.

2. The Problem

We formulate our problem more precisely as follows:

Let there be some number of *groups* A, B, C, etc. Each group has a set of object *types* that are shared by all members of the group. For instance, the types shared by members of group A might be denoted by $a1$, $a2$, etc. We refer to the set of object types used by a group as the *language* of the group. We will be particularly concerned with languages that are *type hierarchies*-- that is, languages in which some object types are specializations of others (their "parents") and automatically inherit properties from their parents.

There are also some number of *translation schemes* for translating types from one language to another. For instance, $T_{XY}(x)$ might be a scheme that translates the types used by group X into types used by group Y. The problem is to formulate translation schemes that (1) preserve the original "meaning" of a given type of object as much as possible while (2) allowing different groups to create or change their type definitions with as much autonomy as possible. Let us explore these two objectives in more detail.

2.1. Preservation of Meaning

We are aware that the meaning of meaning is a complex issue which is studied in a variety of disciplines such as philosophy and linguistics [3, 9, 19]. For our purposes here, however, we ignore most of the complexity and define meaning preservation in the following operational way: The meaning of a message or an object is preserved in a translation if the recipients of the translated object can perform the same range of actions they would have wished to perform had they received and "understood" the original expression. We leave undefined the term "understood" and appeal only to an intuitive sense of what it would mean for receivers to "understand" the original expression. This simplistic definition does not capture, for instance, complexities like what the sender "intends" to convey or how the receiver uses features of the shared context to "understand" the sender's intent. We are not primarily concerned here, however, with the subtleties of natural language understanding. Instead our primary concerns have to do with translating structured (or semistructured) objects into other structured objects that can be processed by a receiver's computer system. For that purpose, the operational definition of meaning we have used seems adequate.

In the context of Lens, for example, we can interpret this criterion as follows. Take the simple scheme T_{AB} that translates any type unknown to a group into the most general type the group has, say Message. Suppose someone in Group A sends to someone in Group B a message of type Action Request, and suppose that Group B does not have the message type Action Request. The operations that the members of the group want to apply on objects of the generic type, Message, such as Reply To,

Forward, etc. are preserved under this scheme. However, the operations specific to Action Request messages such as Making a Commitment would not be available because, for all the system knows, the object received is now an object of type Message no matter what the original group may have intended it to be.

Now suppose further that group B has a message type called Request and that another translation scheme T'_{AB} allows translation of all objects of type Action Request to this type, Request. This translation scheme would preserve the meaning of the original type to the extent that the operations that the group has defined on Request objects overlap with the operations that the group would have defined over the set of objects that the other group classify as Action Request. If the two sets of operations are the same, then the meaning of the type is fully preserved, despite the differences in how the groups name it. If the two sets of operations do not have much in common, then the meaning of the type is not well-preserved. One can imagine situations where this definition is problematic -- e.g., receivers cannot understand the document as its senders did--but is useful enough for our purposes.

One way to ensure full semantic preservation is to require all groups to share the same set of types. However, such a requirement has its costs. The next objective captures the tradeoffs involved.

2.2 Autonomous evolution of group languages

Each group would, in general, like to be able to create or change their type definitions according to their own needs with as few constraints as possible from the other groups. We separate three aspects of this objective: maximizing expressive adequacy, minimizing the need for consensus, and minimizing the need for updates.

2.2.1. Maximize expressive adequacy. Each group would like to be able to express the distinctions that are useful for its purposes. Requiring all groups to share the same set of message types makes it difficult to meet this objective of expressive adequacy because the different goals and contexts of different groups often lead them to make conflicting distinctions. For example, a manufacturing division of a company might want to make many detailed distinctions in their messages about different parts and

subassemblies of many different products while the marketing division of the same company might want to group products in a different way and make detailed distinctions about various market segments.

2.2.2 Minimize need for consensus. Each group would also like to be able to change its type structure while requiring as little consensus as possible from other groups. In general, type structures are not static, but evolve as needs change. The less the need for consensus from other groups, the more easily a group can change its type definitions to meet its current needs.

2.2.3 Minimize need for updates. When a group makes a change, other groups might need to know about the change so they can maintain consistency. For example, if one group renames a type, all the other groups that maintain a dictionary for translating types from that group would need to update their dictionaries. The cost of updates is affected by: (1) the extent to which changes need to be propagated, (2) the complexity of the data structures needed for such propagation, and (3) the extent to which the change requires modification of the existing structures. The less updates a translation scheme requires on these dimensions, the more desirable it would be on this objective.

In the next sections, we will discuss how these different objectives interact when we evaluate different translation schemes against these objectives.

3. The space of possible translation schemes

There are a number of possible translation schemes that achieve, to different degrees, the objectives we discussed above. In this section, we will describe several general solutions to the translation problem, evaluate each solution with respect to the objectives discussed above, and provide specific examples of how each solution could be implemented for the Lens system.

We partition the space of possible translation schemes by using a dimenision that centers around the notion of common language and its relation to group language. Even though there are other ways of categorizing possible solutions, we have found

this dimension useful in placing the schemes that seem practically important. First, we define a *common language* for a set of groups as a language that all the groups in the set use to communicate with one another. In order for all groups to be able to communicate using the common language, each group must either use the common language itself or be able to translate between the common language and the group language.

Given this definition of common language, we characterize the space of possible general solutions to the translation problem in terms of the set theoretic relationships between the group languages and the common language (if any). If we consider only "pure" cases, that is cases where all the group languages have the same relationship to the common language, then, as shown in Figure 3, there are six possibilities. The first possibility is that (1) there is *no common language.* If there is a common language, then it may be (2) *identical* with the group languages, (3) *non-overlapping* with the group languages, (4) a *subset* of the group languages, (5) a *superset* of the group languages, or (6) *partially overlapping* with the group languages.

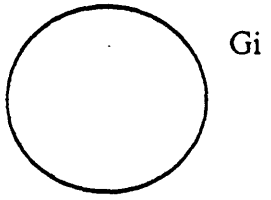
The first four of these possibilities represent interesting practical solutions to our general problem, and we will describe them in the remainder of this section, using the terms (1) *no common language,* (2) *identical group languages,* (3) *external common language,* and (4) *internal common language.* The fifth and sixth possibilities and "hybrid" cases where some group languages have one relationship to the common language and other groups have another relationship can all be analyzed in terms of the first four "pure" possibilities. We discuss these special cases briefly at the end of the section along with several other possibilities.

3.1 No Common Language

In a scheme of this type, there is no common language that is shared by groups that wish to communicate. Instead, each pair of groups must be able to have pairwise translations made into and out of each other's languages. This is, in a sense, the situation that actually prevails in the real world of natural languages such as English, French, and Japanese. In the world of computer-based documents, this scheme implies that there are translating programs for each pair of groups. The most general form of

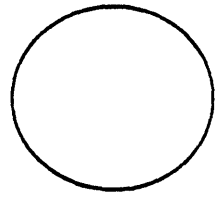
Gi: A Group Language

C: A Common Language



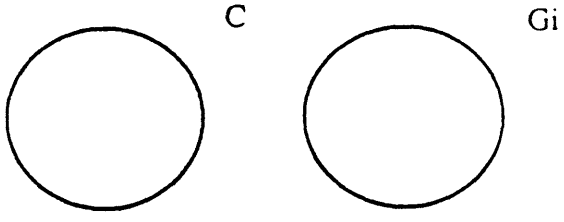
$C = \phi$

(a) No Common Language



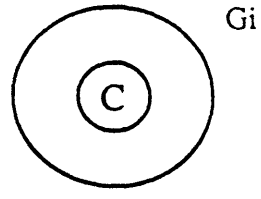
C = Gi for all i

(b) Identical Group Languages



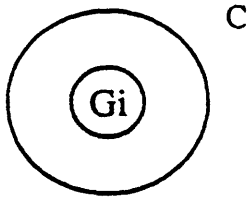
$C \cap Gi = \phi$

(c) External Common Language



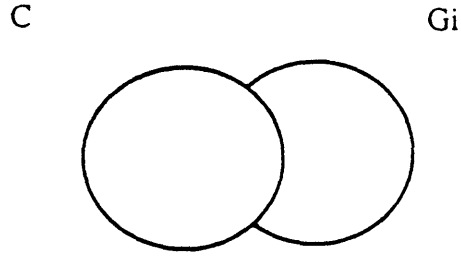
$C \subset Gi$

(d) Internal Common Language



$C \supset Gi$

(e) Superset



$C \cap Gi \neq \phi$, $C \neq Gi$, $C \not\subset Gi$, $C \not\supset Gi$

(f) Intersection

Fig. 3 Possible Relations between Common Language and Group Language

such programs would involve "dictionaries" that provide translation rules for each type from a source language into some type in the target language. Type and map derivations in the Federated Architecture for information sharing [7] illustrates an example of such a dictionary in the context of database integration.

In Object Lens, this solution could be implemented by letting each group have a dictionary for each other group with which it communicates. Whenever an object is received from one of these groups, Lens would use the dictionary to translate the incoming object into a type understood by the receiving group. This translation could involve much more than just changing the type name. It might, for example, include moving and transforming values within the fields of the object. For instance, the following rule might be contained in a dictionary for objects that group B receives from Group A:

IF the object type is GRAUDATE STUDENT
  and the value of SUPPORT field is TA,
THEN convert the object type to INSTRUCTOR,
  and map the fields as follows:
   Student ID -> SSN
   TranslateDPT(DEPARTMENT) -> DPT

The rule says that if the received object is of type GRAUDATE STUDENT and if he is teaching a course, then it should be translated into an object of type INSTRUCTOR after renaming the field 'STUDENT ID' to 'SSN' and 'DEPARTMENT' to DPT', . Furthermore, the function TranslateDPT will be applied to the value of DEPARTMENT in the original message to obtain a value that is understood by Group B (e.g. "Computer Science" to "CS").

Evaluation. This scheme is very flexible in the sense that it can provide case-specific translations in fine-grained details. In this way, the scheme can fully preserve meaning provided that the following condition is met:

If there is a one-to-many mapping from a Group A's type, X, to Group B's types, Yi, then there should be other characteristics of documents of type X that allow unambiguous selection of the Yi to which X should be mapped.

In the above example, the rule that translates Group A's GRADUATE STUDENT type to Group B's INSTRUCTOR is only a partial map because only some of the objects of GRADUATE STUDENT type are mapped to the type in B. To be able to translate all the GRAUDATE STUDENT objects properly, there has to be more than one type in B to which GRADUATE STUDENT is mapped. Hence, there have to be some characteristics in objects that allow the translation scheme to determine which types in B the objects should be translated. In the example, the value TA in the SUPPORT field served as one such characteristic.

This scheme also satisfies some of the objectives of autonomous evolution fairly well. Since the group is not constrained to have any part of its hierarchy shared with other groups, expressive adequacy is not limited. Since setting up translation rules within a group does not affect any other group, no consensus is required. However, a problem with this scheme is that it could be quite costly to set up and maintain. When a group changes its type structure, the change will, in general, have to be propogated to all other groups with which this group communicates so that the other groups can change their translation rules or dictionaries. Each group needs a dictionary for all the other groups with which it communicates. If there are n groups communicating with one another, there would, in general, need to be $n(n-1)$ dictionaries. Also, whenever a new group joins, all the other groups would have to set up a dictionary for that group, and the new group would have to set up dictionaries for all the existing groups.

Applicability. This scheme would be appropriate when the groups already use their own languages and when the efforts to build translation rules and dictionaries for the other groups are justified. Such efforts may be justified, for example, if the number of groups that need to communicate is small or if the groups themselves are stable and their type hierarchies do not change often.

3.2 Identical Group Languages

In this scheme, the common language is the same as any group language. That is, all groups are required to use the same language and communicate through that language. Examples of this scheme include the adoption of global standards such as Dewey Decimal Classification System among libraries or Computing Reviews Classification Scheme used for categorizing literature in computer science. In the context of Object Lens, adopting this scheme means that all the groups share the same object type hierarchy.

Evaluation. In this scheme, meaning is fully preserved because the types and the operations defined on them are the same for all the groups. However, expressive adequacy is quite limited and the need for consensus is quite large. Since all groups have to agree on changes, the only changes that are easy for a group to make will be those that do not affect other groups or those that are valuable to all groups. Even these changes may require a significant overhead to come to agreement. This scheme also requires that all changes be propagated to all groups. However, whereas each group has to deal with the consequences of these changes in the No Common Language scheme, this scheme allows the consequence management to be 'centralized' in that whatever consequences there are can be well thought out once and distributed to each group, For example, one can distribute a translation routine from the old type to the new type together with a type change notice.

Applicability. This scheme would be useful when there are not already well-established groups and when the groups do not differ greatly in their needs. This may occur, for instance, when the domain is very restricted or artificial, or because there is a well-accepted theory about the domain, or simply because the groups share the same goals and environments insofar as they need to communicate. For example, there may be a small set of generic message types such as Request and Commitment that are useful in almost all office environments [25].

3.3 External Common Language

In this scheme, each group uses its own separate language for communication within the group, but there is a single common language for communication between groups. A group communicates with another by first translating its language into the common

language, and then having the receiving group translate the common language into its own group language. In this way, each group only needs to know its own language and how to translate it into and out of the common language. An example would be the adoption of an international language such as Esperanto. Another example can be found in the idea of abstract data type that allows data structures to be implemented in any locally convenient ways as long as they maintain a consistent interface to the outsiders [6, 15].

If Lens were to use this scheme, there would be a common type hierarchy shared by all groups in the sense that each group would know how to translate its own types into the types in this hierarchy and vice versa. For example, suppose that Group A wants to send an object of type, Teaching Assistant, to Group B which does not have that type, but has a type called Instructor. Suppose that the common hierarchy has an object type called Student with an attribute called Support. With appropriate rules set up between the common object and the group objects involved, the Teaching Assistant object can be first translated into an instance of Student with the Support attribute set to TA, which would then be translated to an Instructor object in Group B.

The translation can be done in the same way as described above for the No Common Language scheme--that is, by means of a dictionary containing translation rules. The differences between this scheme and No Common Language scheme is that in this scheme all the groups need to know only how to translate back and forth between their own languages and the common language. So each group only needs two dictionaries, one for translating to and the other for translating from the common language; overall only $2n$ dictionaries are needed for n groups.

Evaluation. This scheme can fully preserve meaning provided that the condition discussed in Section 3.1 is met for translations both into and out of the common language-- that is, when there are one-to-many mappings between types, there must be some other characteristics of the documents that enable the unambiguous selection of a target type. However, often this condition is not satisfied. Hence, in general, the more translations a scheme requires, the less it is able to preserve meaning. Since an External Common Language scheme requires more extra translation steps than the

others we have considered, we expect that it is less likely to preserve meaning than the others.

Since each group can design its own language without being constrained by others, the group languages can presumably have as much expressive adequacy as desired. The need for consensus applies only to deciding what the common language should be and how it should be modified. Any change in a group language need not be propagated to other groups; only the mapping between the changed type and the relevant types in the common language need to be changed. If a type in the common language changes, all the groups that have a local type mapped to that type need to know about the change. But, even in this case, it is only the mappings, not the group languages themseleves, that have to be changed.

Applicability. This scheme would be useful when groups that already use different languages want to communicate and when the cost of setting up translation rules or dictionaries for all the groups is too high. Presumably it is for these reasons that people propose an international language or use English as one. Also, this scheme might be useful when there is a language that is expressive enough, but difficult to use for that reason. For example, we can define a language to be the union of all group languages for the cases where such a union is meaningful. Such a language could express all the distinctions that are of interest to any group, but it might be difficult to use if the types are too fine-grained or place too much cognitive load on the user. Such a language, although not suitable as a group language, might serve as the common language.

3.4 Internal Common Language

For schemes in this class, every group has a part of its language that is shared with all the other groups, and the groups use this shared language to communicate with one another. For example, different academic specialists (say, physicists and anthropologists) may each have their own technical vocabularies, but they can usually communicate with specialists from different fields using ordinary English.

A particularly interesting example of this class arises when the languages are type hierarchies. If all the groups share the root node and other types in the top part of the type hierarchy (i.e., they share the most general types), then all the other types that exist in the separate group languages are specializations of types in the common language. This makes it especially easy to translate group types into the common language, as we illustrate below.

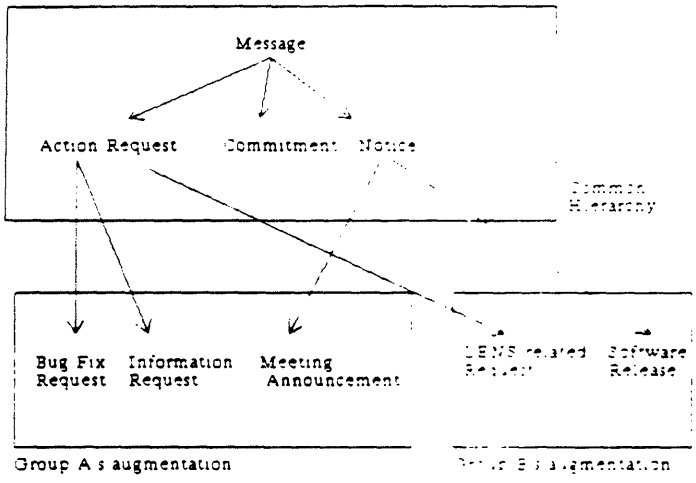


Fig. 4 A Common Hierarchy and its Local Augmentations

We have explored a scheme of this type for Lens called the Local Augmentation Scheme. For instance, all groups might share the message types that are useful almost everywhere, such as Message, Action Request, Commitment, and Notice. Then each group is allowed to locally augment the hierarchy by creating subtypes of these shared types (See Figure 4). When group A sends to group B a message of type, say, Meeting Announcement, the type is automatically translated into the nearest ancestor in the common hierarchy, in this case Notice. Those fields of the type Meeting Announcement (such as Topic) that are also in the nearest common ancestor, Notice, are preserved as they are; those that are not in the ancestor (such as Meeting Time and Meeting Place) are treated as a part of the main text and put in the beginning of the Text field. Of course, if group A sends a message of a type that is already in the common hierarchy, then no translation is necessary.

One way of implementing this scheme is to have each object include the information about its type in the object type hierarchy used by the sending group. If this type is not in the common hierarchy shared by all groups, the object also contains: a) the name of the sending group and b) the nearest "ancestor" of the object type in the common hierarchy. This way, receivers in the sending group can process the message as being of exactly the type intended by the sender, and receivers in all other groups can process it as being of the nearest "ancestor" type in the common language.

Evaluation. This scheme preserves meaning to the extent that the type structure is shared. If the message is of a type in the common hierarchy, then its meaning is fully preserved. If not, then its meaning will be abstracted away to the degree that its ancestor in the common hierarchy preserves its properties. This scheme allows a great latitude in expressive adequacy since each group can add as many types as desired to express distinctions that are important to them. The only constraint on expressive adequacy is that groups cannot completely ignore distinctions made in the common hierarchy. For example, suppose, as before, that the common hierarchy classifies Message objects into the speech act categories such as Action Request, Notice, and Commitment. If a group does not want to use this speech-act based categorization, it can add other children of Message and use them for internal communication, but the group members will still receive messages from other groups based on the speech act categorization.

This scheme only requires consensus among groups on what the common hierarchy should be, and it only requires the propagation of changes in this common hierarchy. The changes made locally within a group need not be propagated to other groups. However, a change in the common hierarchy needs to be propated to all groups. Furthermore, unlike in the External Common Language scheme, such a change might require much change in the local structure. For example, all the groups that have rules or subtypes of the modified type might have to make necessary adjustments.

Applicability. This scheme would be useful when (1) there do not already exist numerous well-established and incompatible language groups, (2) there are important commonalities across groups, and (3) there are also significant local variations among groups. For instance, it seems quite useful in the Lens environment to have a set of

common message types used by all groups (such as Action Request, Commitment, and Meeting Announcement), and to also let each group develop their own specialized message types for their own unique situations. The X.400 standards for electronic messaging are compatible with this approach since they include a field for specifying a message type, and leave open the possibility of different groups developing their own conventions about how this field is used.

3.5 Common Language as a Superset of Group Languages

In this scheme, the common language is a superset of the group languages. An interesting instance of this scheme is the following: Suppose that different groups maintain different type hierarchies. However, when a group receives an object of unknown type, that type and any of its supertypes that are not known locally are automatically imported into the local type hierarchy. In this case, the common language is the union of all the groups' type hierarchies, because any object in the union can be used to communicate among the groups.

Evaluation. In this scheme, meaning can be fully preserved if the types are imported together with all their operations. However, doing so can be costly; or if the operations are written in a programming language not supported in the receiving group, automatic importation can be impossible. This scheme allows the groups to express whatever they want without much need for consensus. However, this scheme does not address the problem of how the imported objects are related to the local types. It also has the potential problem of creating different types with the same name, thus "cluttering" the type space. Also, in this scheme, the update propagation can be costly if we want to maintain consistency between the imported type and the original. Any change made to a type would have to be either broadcast or sent to all the groups who imported the type. The latter requires that whenever a group imports a type, the originating group would have to be notified.

Applicability. In spite of the potential costs of this scheme, it is simple and offers a large degree of autonomy for groups. It can be practically useful when the costs of importing foreign objects is not too high. For example, if only a small number of people are involved, updates may be rare (and their total cost low) and there may be

little danger of cluttering the type space. The usefulness of this scheme is also increased when the object types are hierarchically arranged and some types are known to both groups. In this case, a form of the local augmentation scheme can be used to apply operations from a shared supertypes to newly imported types.

3.6 Other translation schemes

We have discussed above five of the six set-theoretic possibilities (Figure 3) for the relationship between the common language and a group language. We can also have (a) hybrid cases, where some groups have one relationship with the common language and other groups have different relationships, and (b) sequential composition of the basic cases. We show elsewhere [14] that these other possibilities, as well as the sixth possibility not discussed (Intersection in Figure 3), can all be analyzed as combinations of the five schemes discussed above. In [14], we also discuss a translation scheme that exploits the subsumption property of certain type hierarchies. In this scheme, a message is automatically classified into the type hierarchy according to what fields it contains [20]. This scheme requires no explicit agreement about the types used, but it does require agreements about the field names themselves as well as some other restrictions on the form of the type hierarchy.

3.6 Summary of translation schemes

Table 1 summarizes the different classes of translation schemes we have considered. The detailed discussions above are summarized in the table by rough qualitative rankings of the schemes on each objective. In each column, the schemes that in general satisfy the objective better receive more stars. For example, the schemes that receive *more* stars in the column, Need for Consensus, have *less* need for consensus.

As we have seen, there is a complex tradeoff between the objectives we are trying to achieve. For example, the pairwise translation scheme used when there is no common language allows full expressive adequacy and little need for consensus, but at the cost of more need for propagation of changes. An identical group languages scheme has full preservation of meaning, but much less expressive power. And any scheme that uses trivial translation rules (e.g., all foreign message types are translated into generic

Messages) can reduce the need for consensus and for propagation of change at the expense of preservation of meaning.

Table 1 Evaluation of the different classes of translation schemes.

	Meaning Preservation	Expressive Adequacy	Need for Consensus	Need for Updates
No Common Language	**	*****	*****	*
Identical Group Languages	*****	*	*	***
External Common Language	*	***	***	****
Internal Common Language	***	**	**	**
Superset	***	****	****	*

We have shown how the classes of translation schemes we have discussed exhaustively partition the space of possible solutions. However, the examples of these schemes that we considered are by no means the only possible ones. For instance, there may be other schemes that exploit special properties of particular languages just as the Local Augmentation Scheme exploits the inheritance property of type hierarchies.

4. An Attractive Composite Scheme: Partially Shared Views

In this section, we present a composite scheme, called Partially Shared Views (PSV), that combines some of the best features of the basic schemes discussed above: It allows groups to share some object types, to set up translation rules for other object types, and to use default translations into shared "parent" types for still others.

This scheme exploits the notion of views to modularize related type definitions and make relations among such modules explicit [1, 5]. A view is a set of object types and their relations. A view V_2 is a subtype of V_1 if some of the message types in V_2 are specializations of ("children" of) the message types in V_1. Figure 5 shows a few examples of views.

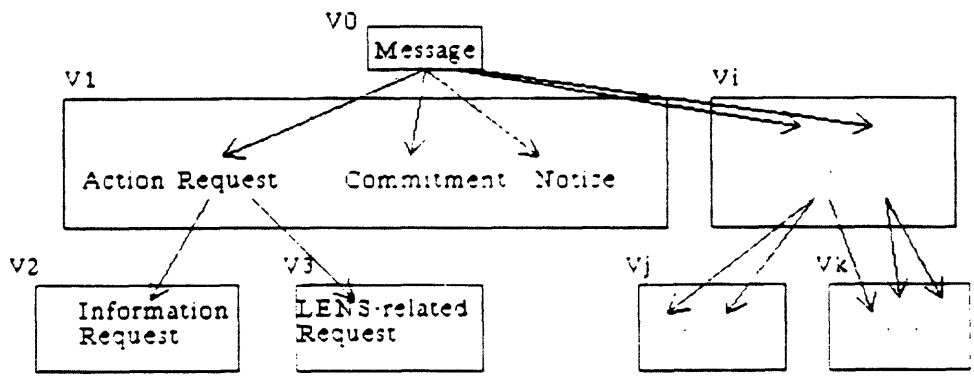


Fig. 5 Examples of Views

4.1 How It Works

When a group G receives an object from another group, then one of the following happens:

(1) If the object belongs to a view that has been 'adopted' by the group, then the object is treated as being of its original type.

(2) If the object is from a view that has not been adopted by the group, then:

(a) if the object meets a pre-specified criterion (e.g. if it is from the system administrator), then the view that it belongs to is automatically adopted and the object is treated as in (1).

(b) if the group has a translation rule set up for the object's type, then the object is translated according to the rule.

(c) if the group has a translation rule set up for a supertype of the object (but not for the object type itself), the object is treated as the supertype and translated accordingly. That is, all the fields that belong to the supertype get translated as specified in the translation rule, and all the fields that are not included in the supertype are simply added to the beginning of the comment field of the object.

(d) Otherwise (if none of the above conditions holds), the object is automatically translated into the most specific supertype in the most specific view that is shared by the sending group and the receiving group. When the translated object is shown, the information about its original type and view is preserved so that the receiving group can, if it wishes, set up translation rules for this new type or, possibly, adopt the entire view to which it belongs.

The above algorithm is based on the following assumptions:

1. All groups share the default view, V0, which consists of the most general type (e.g. Thing).
2. When a group adopts a view, it also adopts all the views which are its ancestors.
3. Each object includes the following information:
    a. the type of the object, t.
    b. a globally unique name of the view within which this type is defined, v.
    c. the "genealogy" of the object, such as $\{(t_1, v_1), .., (t_n, v_n)\}$, where the v_i's are the names of ancestor views of v ordered from most specific to more general, and each t_i is the most specific ancestor type of t within v_i.

The actual algorithm used is actually a bit more complex than presented here because multiple parents are possible. In these cases, the genealogy information has to be presented as a nested list and we have to make sure that we deal with multiple parents appropriately. But, for simplicity, we assume below that each object has only a single parent.

4.2 Illustration

We illustrate the PSV scheme by describing its implementation in the Object Lens system [10]. Object Lens and PSV are implemented in Interlisp-D (and Loops) on Xerox 1100 series workstations connected by an Ethernet. All the features described below have been implemented and have received limited testing at our site.

The following scenario illustrates how PSV works. This hypothetical scenario is inspired by our experience with test sites for the Information Lens, and illustrates the use of PSV for translating messages only. (Other kinds of objects are not included in this scenario). Site A is loosely based on our own group at MIT and Site B is loosely based on our primary external test site. Suppose there are five views related as shown in Figure 4.1. The Base View is the parent of all the other views; it contains only the most general message type: Message. The Standard View is intended to contain a set of templates that would be useful for almost any group using Lens. Here, we illustrate one possible such set by three message types based on speech acts (Action Requests, Notices, and Commitments) and several specializations of these speech acts such as Meeting Proposals and Meeting Announcements. Site A's view adds to the Standard View several specialized message types that are of use in that particular group (e.g., Bug Fix Requests and Bug Fix Announcements). Figure 4.2a shows all the templates used by Site A. The templates that belong to the Site A View itself are shown in blackshade, and the templates belonging to the Base View and the Standard View, are shown in grayshade. Site B's view, as shown in Figure 4.2b, is a different specialization of the Standard View. Site C's view, even though not shown here in detail, is included to demonstrate that no site is required to adopt the Standard View. For instance, Site C might use some basis for message type categorization that is different from the speech act scheme.

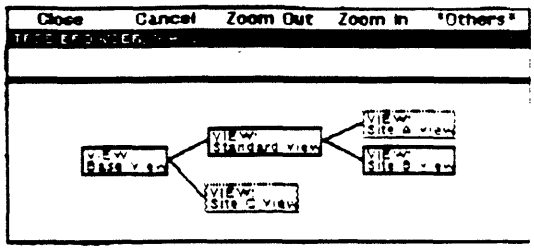
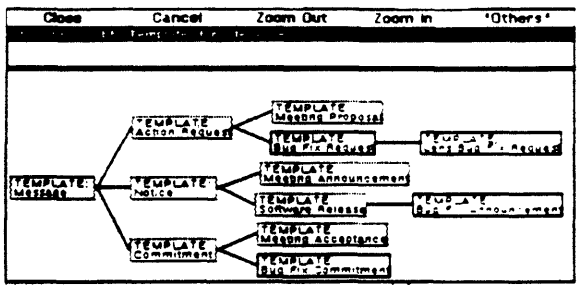
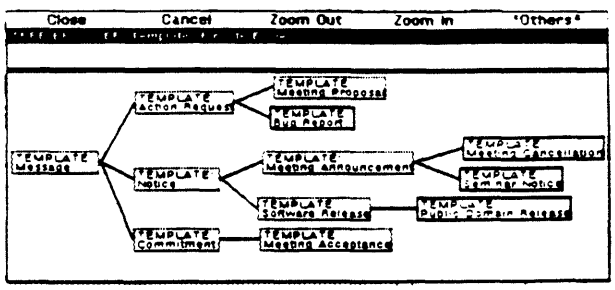


Fig. 6 A sample set of views.



(a) Site A



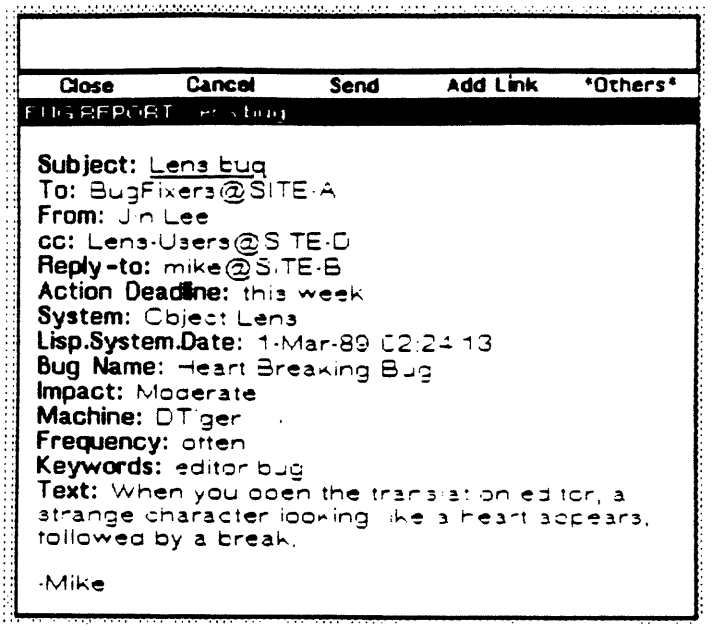
b) Site B

Fig. 7 Templates in the views adopted by Sites A and B, respectively.

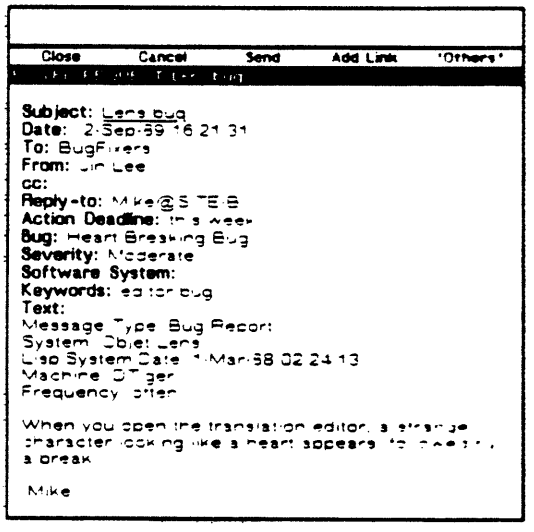
4.2.1 Explicit translation rules

Suppose that someone at Site B sends the message of type Bug Report shown in Figure 8a to Site A. We further assume that Site A has the following translation rule set up for this type:

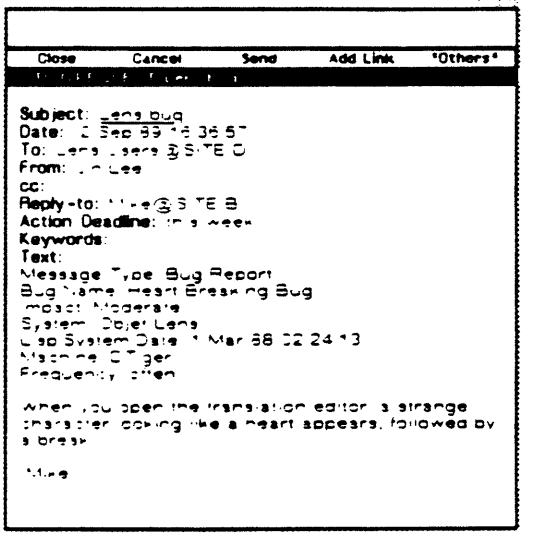
IF the message is of type Bug Report in Site B View
 and the value of the field System is Lens,
THEN change its type to Bug Fix Request in Site A View
 and use the value of Bug Name for Bug
 and the value of Impact for Severity.



(a) The original message



(b) Translation to Site A's view an explicit translation



(c) Default translation for Site based on D into the nearest shared rule "ancestor" type

Fig. 8 A Sample message and its translation into the views used at two other sites.

This rule causes the Bug Report shown in Figure 8a to be translated into the Bug Fix Request shown in Figure 8b. Furthermore, all the operations applicable to the translated type are applied to the message. For example, if the receiver at Site A has set up a rule to automatically archive all Bug Fix Requests into a database, then this message will be so archived.

4.2.2 Default Translation

Now suppose that the original message in Figure 8a is also received by a Lens user at Site D, and suppose further that Site D has adopted only the Standard View with no local augmentation of its own and no explicit translation rules. Figure 8c shows how the original message would be automatically translated by default into the nearest shared "ancestor" type used by both Sites B and D. In this case, the nearest shared ancestor type is Action Request, so the message is translated into an Action Request and the contents of the Action Request fields (including Action Deadline) are carried over into the new message. All the other fields from the original message are simply inserted in the text field of the new message. When users at Site D receive this message, any rules they have set up for Action Requests will be applied to it.

4.2.3 Maintaining views and translation rules

We expect that, in most cases, the adopting of views and the creation of translation rules will be done by the same "system administrators" who maintain each local group's message types. In these cases, most users would just load new versions of their system when announced by their local system administrator and they would not even need to be aware of what or how views were being used. Some users, however, might have communication needs that differ significantly from others in their local group. For instance, they might communicate frequently with people at another site who use a number of specialized message types for which the local system adminstrator has not created any specialized translation rules. In these cases, the individual users might want to adopt another view or create their own translation rules, and the PSV scheme allows them to do this quite easily.
a to Site A. We further assume that Site A has the following translation rule set up for this type:

4.2.4 Implementation issues

In our prototype implementation of this scheme, we did not attempt to deal with all the issues that would arise in implementing it on a large scale. In this section, we briefly sketch a few of these issues and possible solutions:

(1) *User interface for translation rules.* In our prototype implementation, the user interface for specifying translation rules was relatively primitive. There are a variety of attractive possibilities for a more graceful interface, however. In the Object Lens environment, for example, it would be consistent to have translation rules defined as specializations of the generic rules used by all agents. These rules could specify in their IF conditions, the criteria for messages to be automatically translated, and their THEN actions could contain a set of "mapping" objects, each one specifying a source field (from the original message), a destination field (in the translated message), and an optional conversion function to be aplied to the field contents. An alternative approach for specifying translation rules would be to let the user "draw" connections between the fields of the source object type and the destination object type using "rubber banding" lines. This approach was used, for example, in the KEEconnection system for importing information from remote databases into knowledge-based systems [8].

(2) *Storing and transmitting view definitions.* In a large scale implementation of a scheme like this, a critical issue is where and how views (i.e. sets of object type definitions) would be stored and transmitted. We believe it would be desirable to have two options: (a) Users who have convenient access to shared file servers would presumably like to be able to store and retrieve views using these file servers. (b) Users who do not share access to file servers should also be able to simply mail view definitions to each other.

(3) *Including type names and "genealogy" in objects.* We believe it is desirable to have objects be "self-contained" in the sense that any receiver can tell, by looking at the object alone, how to interpret it (i.e., whether this is a view and type already known to the receiver). It is also desirable, especially in the case of message objects, for the objects to be readable by human receivers who are not users of the PSV scheme. These

two objectives present a slight conflict since including globally unique identifiers for types, views, and type genealogies will require including some "ugly" identifying information in the human readable objects. We believe, however, that in most cases a satisfactory solution will be to compactly encode this information in a character string that is "hidden" somewhere in the object. For example, with messages, this string can be either (a) placed in a Message Type field in the message header (analogous to the Message Identifiers used by many current mail systems), or (b) "buried" at the very end of the message after a human readable label like "Identifying information:".

4.3 Evaluation

PSV is a composite of several of the basic schemes discussed in Section 2. As noted above, between groups that share the same view, the scheme is an instance of the Identical Group Language scheme. To the extent that the groups have translation rules between views, it serves as an instance of the No Common Language scheme. To the extent that one automatically adopts views, it is an instance of the Superset Common Language scheme. PSV, however, provides a finer control over automatic adoption by allowing users to specify the conditions under which a view should be automatically adopted. When the groups share only some views and do not yet have translation rules set up between the views not shared, the scheme acts as an instance of the Internal Common Language scheme. The scheme thereby allows finer-grained adoption of translation methods among different groups while still providing a unifying framework. Within this framework, a group can weigh the importance of different objectives in different cases and select a translation method appropriate for each case. For example, if it is quite important for a group to understand another group's language, then it can decide to share the same view or adopt a view where translation rules can be set up precisely. On the other hand, if such precise understanding is not important enough to justify the effort, the group can create its own view and rely on the default translation of the type into its nearest ancestor in the shared view.

4.4 Applicability

Since the Partially Shared Views scheme combines many of the best features of the other schemes it is potentially relevant in very many situations. In a sense, each of the other schemes is a special case of this one. This scheme, therefore, may require somewhat greater implementation complexity, but it allows a great flexibility in using a combination of different translation schemes for different situations.

For instance, this approach seems clearly desirable for the Lens system. For similar reasons, the approach also seems desirable for many other office systems. We describe below, in Section 6, how the Local Augmentation scheme could let systems using different subsets of EDI (Electronic Data Interchange) forms achieve a certain degree of interoperability by agreeing on generic global standards that leave room for augmentation by different subgroups. The Partially Shared Views scheme seems even more appropriate in these situations because it allows many overlapping subgroups to use their own standards for communication with members of the same subgroups, and also to set up translation rules for communication with other subgroups, even when there are no significant global standards.

5. Other Applications

In this section, we illustrate how our scheme can be applied to another problem that requires exchanging information between groups that use different types of data: the problem of exchanging electronic business forms between organizations (Electronic Data Interchange, or EDI). Elsewhere, we also show how PSV can be applied to the problem of integrating heterogeneous databases [12].

An increasing number of companies are now using electronic connections between independent organizations to exchange certain standard business forms (such as purchase orders and invoices) [23]. This kind of exchange (called Electronic Data Interchange or EDI) is greatly facilitated by common formatting standards for the forms, and many industries have begun to develop such standards. For example, there are now ANSI standards for generic forms such as Purchase Orders, Purchase Order

Change Requests, and Invoices, and industry specific forms such as Shipment Information fo Air Carriers, for Motor Carriers, and for Ocean Carriers.

As currently used, these different types of forms constitute an essentially "flat" space, with no notion of specialization or inheritance, but it appears that the ideas about type inheritance and incremental sharing we discussed above could be of use in EDI systems. For example, if EDI forms are defined hierarchically, then it would be easier to add new types of forms that shared many of the characteristics of previous forms. Shipment information forms for a new transportation medium (like bicycle couriers, for instance) could be added by specializing a generic Shipment Information form already present. More interestingly, certain automatic translations between forms would be possible without anyone needing to explicitly define them. For instance, if a change of plans required an air shipment to be re-routed by truck, most of the fields for the new shipment information forms could be filled in automatically from the old ones. In some cases, this might be all that is needed. If not, explicit translation rules could be created or, in special cases, people could complete the translation manually.

To illustrate more concretely how these ideas might be implemented, Figure 9 shows a possible organization of the EDI forms (called "Transaction Sets") for various kinds of shipment information. The "leaves" of this hierarchical organization contain all the data elements in the currently defined standards for the different types of shipment information forms [23]. The figure also shows how some new generic forms can be defined at various levels of abstraction to take advantage of the type inheritance mechanism we have proposed here.

7. Conclusion

In this paper, we have considered a fundamental problem of coordination: How can groups that use different languages communicate with each other? This problem arises repeatedly in human organizations and also, increasingly, in distributed computational systems. We have characterized and evaluated a set of generic solutions to this general problem and illustrated their application in two practical contexts: the Object Lens cooperative work system, and the use of Electronic Data Interchange standards. In particular, we described a composite scheme called

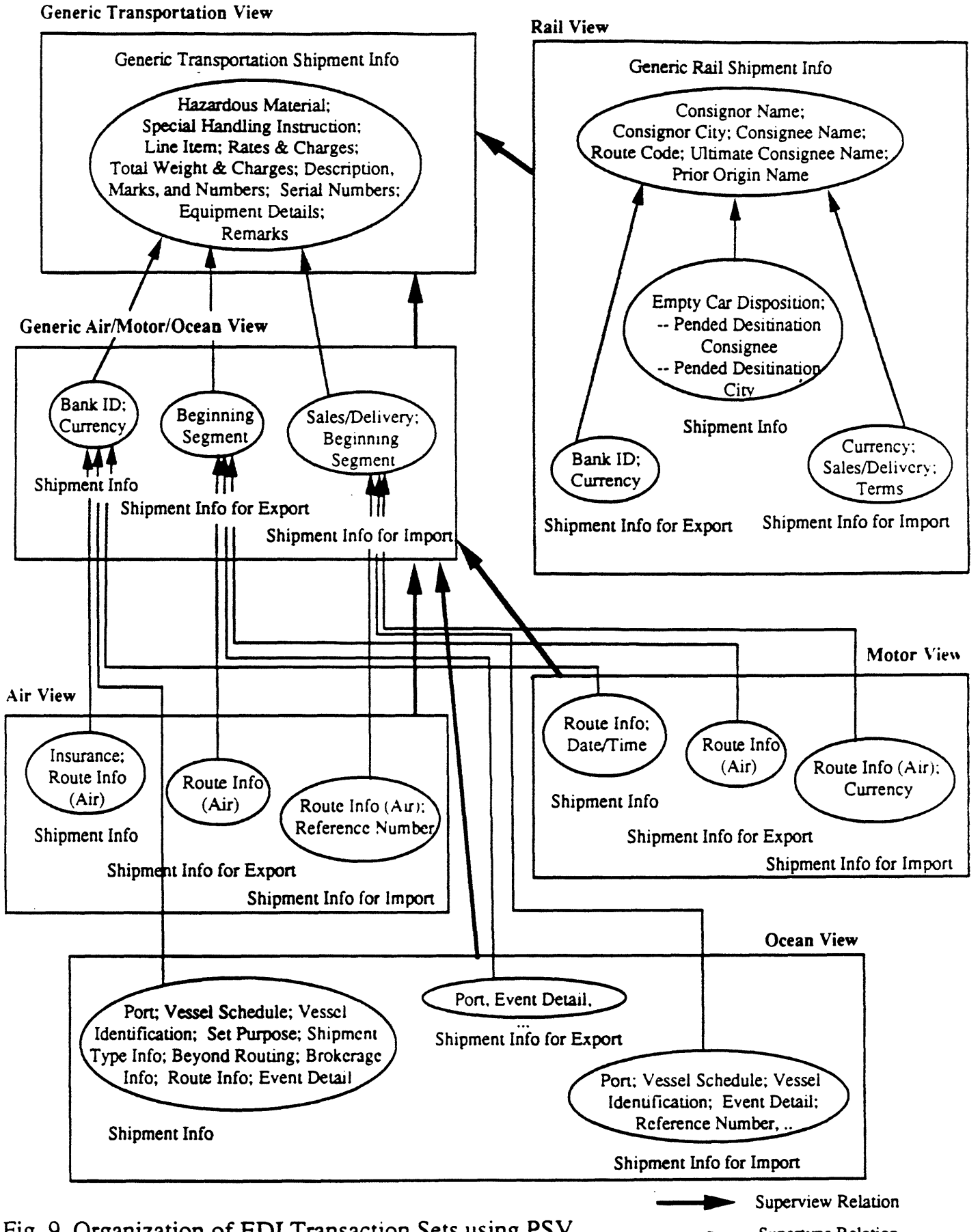


Fig. 9 Organization of EDI Transaction Sets using PSV

Partially Shared Views that combines many of the best features of all the schemes we analyzed. A key insight of our analysis is that when groups communicate using objects that are arranged in a specialization hierarchy, they can take advantage of a powerful form of automatic translation: "foreign" object types can be automatically translated into their nearest common "ancestor" types and thus still retain much of their original "meaning." The partial interoperability attained automatically in this way makes possible a much more flexible kind of standard. No longer does everyone have to agree about all aspects of a standard; they can still get substantial benefits from whatever agreements they do have.

ACKNOWLEDGEMENTS

We thank Kevin Crowston, Kum-Yew Lai, Wendy MacKay, Jim Miller, and David Rosenblitt for their comments that helped identify and elucidate problems that we otherwise would have not seen. This research was supported in part by Digital Equipment Corp.; the National Science Foundation; Wang Laboratories, Inc; Xerox Corporation; General Motors; and Bankers Trust Company.

REFERENCES

1. Barber, G. Office Semantics.. Ph.D. thesis. Massachusetts Institute of Technology, 1982.

2. Batini, C., Lanzerini, M. & Navathe, S. A comparative analysis for methodologies for Database Schema Integration. ACM Computing Surveys v. 18(4). 1986

3. Barwise, J. & Perry, J. *Situations and Attitudes* MIT Press. Cambridge, MA. 1983

4. Fikes, R. & Kehler, T. The role of frame-based representation in reasoning. In *Comm. ACM* v.28(9) September. 1985 pp.904-920

5. Goldstein, I.P. & Bobrow, D.G. Layered networks as a tool for software development. *Proc. 7th Int'l Conf. on Artificial Intelligence*, 1981.

6. Guttag, J.V. 1977. Abstract Data types and the development of data structures. Comm. ACM, v.20(6). June 1977, pp.396-404

7. Heimbigner, D. & McLeod, D. A federated architecture for information management. In *ACM Trans. Office Information Systems* 3(3) July, 1985

8. IntelliCorp Inc. KEEconnection^{tm}: a bridge between databases and knowledge bases. An IntelliCorp Technical Article. Moutain View, CA. 1987

9. Jackendoff, R. *Semantics and Cognition* MIT Press. Cambridge,MA. 1983

10. Lai, K.Y., Malone, T.W., & Yu, K.C. Object Lens: A 'spreadsheet' for cooperative work. *ACM Transactions on Office Information Systems* 6:332-353 1988

11. Lee, J. Knowledge base integration: what can we learn from database integration research? MIT AI Memo 1011. 1988

12. Lee, J. & Malone, T.W. How can groups communicate when they use different languages? Translating between Partially Shared Type Hierarchies. Sloan WP #. 3076-89-MS. the extended version of [13] MIT. September 1989

13. Lee, J. & Malone, T.W. How can groups communicate when they use different languages? Proc. Conference on Office Information System, pp.22-29. Palo Alto, CA. 1988

14. Lee, J. & Malone, T.W. Translating type hierachies: framework analysis and a proposal. Sloan WP #. 1974-88. M.I.T. Jan 1988

15. Liskov, B. & Zilles, S. Programming with abstract datatypes. *SIGPLAN Notices* MIT Press. Cambridge, M.A. 1974

16. Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A., Cohen, M. Intelligent information-sharing systems In *Comm. ACM* v.30(5) May, 1987 pp.390-402

17. Malone, T.W., Grant, K.R., Lai, K.Y., Rao, R., & Rosenblitt, D. Semi-structured messages are surprisingly useful for computer supported coordination. In *Trans. on Office Information System* v.5(2) 1987 pp.115-131

18. Nii, P. The blackboard model of problem solving. *The AI Magazine.* Spring, pp.38-53

19. Putnam, H. The meaning of 'meaning'. In *Language, Mind, and Knowledge*, ed. by K. Gunderson. Minneapolis: Univ. of Minnesota Press. 1975

20. Schmolze, J.G. & Lipkis, T. Classification in the KL-ONE knowledge representation system. *Proc. 6th Int'l Joint Conf. on Artificial Intelligence*, 1983.

21. Skarra, A.H. & Zdonick, S.B. The management of changing types in an object-oriented database. In *Proc. OOPSLA*, September 1986. pp. 483-495.

22. Stefik, M., Bobrow, D.G., Mittal, S. & Conway, L. Knowledge programming in LOOPS: Report on an experimental course. *The AI Magazine*, pp.3-13, Fall 1983

23. TDCC: The Electronic Data Interchange Association. The United States Electronic Data Interchange (EDI) Standards. V.4 Data Segments and Data Elements. Washington, D.C. 1988

24. Tsichritzis, D. Form management. In *Comm. ACM* v.25(7) July pp.453-478

25. Winograd, T. & Flores, F. Understanding Computers and Cognition: A New Foundation for Design. Ablex, Norwood, N.J. 1986