A Conceptual Model of Groupware

Clarence (Skip) Ellis
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430, USA
E-mail: skip@cs.colorado.edu
Tel: +1-303-492-5984

Jacques Wainer
Department of Computer Science
State University of Campinas
Campinas, SP 13081-970, Brazil
E-mail: wainer@dcc.unicamp.br
Tel: +55-192-39-3115

June 1994

ABSTRACT
This paper discusses a conceptual model of groupware consisting of three complementary components or models: a description of the objects and operations on these objects available in the system; a description of the activities (and their orderings) that the users of the system can perform; and a description of the interface of users with the system, and with other users.

KEYWORDS: Groupware, CSCW, collaboration technology, system modelling, ontological model, coordination model, user interface model.

INTRODUCTION

This paper is concerned with technology mediated workgroup systems, also sometimes called CSCW systems or groupware. We take the broad definition of these terms, as presented by Schmidt and Bannon [16]. We present a conceptual model which is applicable to a wide range of groupware—from electronic meeting rooms, through video conferencing, to workflow systems. The issues raised by the model are sometimes similar to, but frequently distinct from single user systems issues. This is especially true within the user interface domain.

How can one compare two distinct groupware system? How can one describe a particular system? These questions seem to be related to the question of what characterizes a groupware system: if one knows what characterizes a system one can describe it and compare it to others. This paper proposes that, from the point of view of the user (or users) of a groupware system, one can characterize such a system in three complementary aspects, or models as we will call them in this paper. The three aspects are:

- a description of the objects and the operations on these objects that are available to users;
- a description of the dynamic aspects of the system: the control and data flow;
- a description of the interface between the system and the users, and amongst users.

We call these three aspects the ontological model, the coordination model and the user interface model respectively.

We must stress that these three models describe the system from its users' point of view. There are aspects of the system that are not captured by this three model approach - for example implementation aspects which are mainly of concern to the implementor. Thus, whether a system has a centralized or a distributed implementation only indirectly affects the users, and such distinction is not captured by our three model approach. However, distribution is a major issue from the implementation point of view.

The three-models approach is more than just a description of the system from the users' perspective, or for the users' benefit. We believe it corresponds to a functional description of the system, and it reflects the designer's intuition of the important aspects of the system. In this sense, the three conceptual models have strong similarities with system specification concepts in software engineering. The ontological model is related to static specifications of systems like, for example, entity-relationship diagrams and object-oriented analysis. The coordination model would correspond to functional specifications, data-flow diagrams, Petri-nets, or other forms of dynamic description of systems.

But there are some differences between the ontological and coordination models, and the static and dynamic models of software in general. The ontological model is

a static description of the objects *available to the users for manipulation*, thus it may not describe aspects of the implementation of the system itself. The difference between the coordination model and , for example, a data-flow diagram is in the same line as above: the coordination model describes the organization of activities *to be performed by the users*, and not the organization of processes to be performed by the system. Thus both models refer to the users' view of the system and not the implementation view. Furthermore, the essence of this proposal is not the realization that one can distinguish a static and a dynamic aspect of a groupware system, but the *concepts and definitions that derive from this distinction*. For example, we will discuss below the concept of operational and intended semantics of objects, which derive from a static description of the system. We will also discuss levels of simultaneity and currency which are derived from the dynamic component of the system.

We believe that this work's main contribution to the CSCW community is that it provides some concepts and definitions that can be very useful in describing and designing groupware.

ONTOLOGICAL MODEL

The ontological model of a groupware system is the description of the classes of objects and the operations that the system provides to the users. In fact, the concept of an ontological model is by no means limited to a groupware system; all (or nearly all) interactive systems, be they single-user or groupware systems, embody some ontological model. For example, a group drawing tool differs from a VLSI CAD tool by the ontological model it embodies. A drawing tool offers objects like straight lines, points, curved lines, closed regions, colors and textures, and operations to create and modify these objects. A VLSI CAD tool offers objects like transistors, resistors, terminals, crossings, and so on, and the corresponding operations to create, move, modify and set attribute values on these objects. Even though a terminal in the VLSI CAD system may appear as a rectangle it should not be confused with a rectangle object in a drawing tool. The terminal object has different attributes from the rectangle object, and it is subject to different operations.

Similarly, two groupware systems may be within the same general category (e.g. both review systems or both electronic mail systems), but they may embody different ontological models. Let us compare two review systems CA-ForComments [3] and PREP [13] under this view. They both are concerned with text based documents, and offer the usual edit operations on them. Both systems also define the concept of comments that are contributed by the reviewers.

CA-ForComments offers the objects "comment," "revision," and "dialog" (among others), which are all kinds of text but intended for different uses. The comment object is supposed to contain the reviewer's comment on a sequence of lines of the original document. Thus, besides operations for creating and editing of a comment, ForComments offers an operation for attaching a comment to a sequence of lines in the original text. The revision is supposed to contain a replacement text for a sequence of lines, and besides the creation and attachment operations, the system provides an operation for replacing the text of the original document with the text of the review object attached to it. The object dialog is a comment on a comment, and can only be attached to another comment.

On the other hand, PREP (the version described in [13]) provides just a generalized comment object that can only be attached to paragraphs of the original text. These generalized comments may contain comments in ForComment's sense, but may also contain text with other intended meaning. In fact, [13] suggests a generalized comment called "plan" that is created and attached to paragraphs by the author of the document and describes the author's intention for each paragraph.

Thus, although both systems are designed with the same general purpose of facilitating the task of a group of people in reviewing a document, they have quite different ontological models. Notice that ForComments attaches comments to sequences of lines, while PREP attaches comments to paragraphs. Of course these differences are a reflection of the respective designer's different intuitions on what are the "best" object types and operations to accomplish the task of reviewing a document by friendly reviewers. It is very likely that a system to aid the reviewing of documents by not so friendly reviewers, for example revision of a technical paper submitted to publication to a scientific journal, would embody yet a different ontological model.

The description of the two review systems above also provides insights on important aspects of an ontological model, described below.

Some Definitions

The main components of the ontological model are objects and operations. Objects are the data structures upon which the participants (i.e. users) operate; operations on objects are an important aspect of a participant's contribution to the job. For example, in a

group drawing system part of a participant's work is a sequence of operations on objects, like creating a polygon, changing an attribute of a circle so that its interior has some color, deleting a line and so on. In a workflow system, part of the participant's work can be seen as a sequence of editing operations on objects such as forms or fields or documents. Another important part of the participants' work is interacting with group members and the environment for problem solving, decision making, group membership maintenance, etc. In groupware systems and models, there is much more concern with these aspects of work than in single-user systems.

The objects are modeled using the standard concepts of attributes and values: objects have attributes and these attributes may have one or more values. Values can be either atomic or other objects. Although we use the term object, this modeling does not strongly relate to the concepts of object-oriented methodologies. In particular we will not use the concepts of methods, message passing, and inheritance. We use the term "object" as a synonym of "data structure."

Operations are transformations that act on the objects. We will broadly classify the operations as **view operations**, which allow a user to know about the existence of an object and possibly to know the value of its attributes; **create operations**, which create new instances of an object; **modify operations**, which change or add values to some of the attributes of an object instance; and **destroy operations**.

It is also important to notice the distinction between object classes, which represent generic, prototypical objects, and object instances, which are the entities (members of classes) upon which the operations can be applied.

Finally, we will also speak of relations between or among objects. We will say that some object instance stands in some relation with another object instance, or that some instance is linked to another through some relation. The concept of relation is secondary since it can be implemented using the concepts of object and attributes. For example, a binary relation can be implemented as a new class of objects, and to state that object A and B stand in that relation to each other correspond to the creation of a new instance of the relation objects, with A and B as the values for the "argument" attribute.

Intended and Operational Semantics

When talking about objects classes is important to bear in mind the distinction between intended and operational semantics of the objects. The **intended semantics** of an object class is the description of the intended use of the instances of that class. For example, in CA-ForComments, the class "comment" is intended to contain a text that expresses the reviewer's opinion about a sequence of lines, while the class "revision" is intended to contain a replacement text. Similarly, in PREP, the intended semantics of the class "plan" is to contain text in which the author comments on his/her intention for the particular paragraph to which the plan is attached.

The intended semantics of the classes of objects provided by the system is usually explained in system's user manuals, and is the result of the designer's intuition or analysis of what are the concepts that would make solving the task easier.

The **operational semantics** of a class are the constraints on the possible relations between this class and the other classes of objects in the systems, and the set of operations that can be applied to instances of this class. Thus the operational semantics of the class revision in ForComments is that it can only be attached to a sequence of lines of the original document, and nothing else. The operation "replace" when applied to a revision would replace the lines of text to which the revision is attached by the contents of the revision. Similarly, a comment can also be attached to a sequence of text lines, and objects of the class dialog can be further attached to comments (dialogs are comments to comments in ForComments). The operation replace cannot be applied to comments.

From the point of view of the operational semantics, there is no distinction between the various classes of objects in the systems. They are all described in terms of possible operations and possible relations among them. But from the point of view of intended semantics, the classes can be classified in object-level classes, like text or sequence of lines, and meta-level classes, like comment or plan, and possible meta-meta level classes, like dialogs.

In some cases the ontological model, and in particular the intended semantics of some classes of objects, are the main concept behind the groupware system, and its major contribution. A clear example of this are the Issue Based Information System (IBIS) [4, 15]. The main idea behind IBIS is that chunks of knowledge in a group discussion should fall into only three categories, and that these categories relate to each other in few and well defined ways. "Issues" are points that need resolving, or questions that need answering; "positions" are possible ways of solving an issue; and "arguments" are statements that either support or go against a position. The main relationships among these object classes are that a position answers an issue, and an argument

supports or opposes a position.

From the point of view of the system which implements the ontological model, the intended semantic of the objects is not implementable: there is no way the system can check if an instance of an object is used according to its intended semantics. For example, it is not possible with current technology to expect that an IBIS system could enforce the fact that a position has to contain a possible solution to the issue to which it is attached. For that to be possible, the system would have to understand the meaning of the issue and check that the content of the position is indeed a possible answer to that issue. Thus, the system cannot enforce the intended semantic of these objects, but it enforces their operational semantic, for example by not allowing the creation of a supports link between two positions, and so on. Thus, in an IBIS, the operational semantics of the objects is the descriptions of what are the possible links between the classes of objects, whereas the intended semantics is what the objects in each class should mean to the users.

Similarly, one of the important aspects of the Coordinator [17] is the classes of messages it defines, or more specifically the intended semantics of those classes, and the relation between the classes. But, once again, the system cannot enforce this intended semantic.

Other Aspects of the Ontological Model

Operation rights and access rights

Another concept related to the ontological model are the rights to perform operations or just to access an object (or the content of an object). A user, at a certain moment, may have the right to access an object, that is, to view its contents, but may not have modify rights to perform modify operations on it. At another moment, the same user may have access rights to the object and rights to perform only some of the operations but not all, and so on. We will see later that these moments where the users have different rights are usually associated with activities, which is a fundamental concept of the coordination model.

Groupware systems typically have an expanded set of rights which must be considered; privacy, protection, and control [5] are all important considerations within the ontological model

Automatically generated values

The final concept of ontological models is based on the fact that not all values of an attribute of an object are set by the participants, but some are automatically set by the system. The most clear example of automatically generated values are time and user stamps. For instance, an e-mail system could tag each message created with user-identification, and the date stamp. Usually, the system that implements automatically generated values will provide operations that are able to access, but not modify these fields. Thus in the e-mail system above one could list all messages sent by a particular person, or all messages sent after a certain date.

Summary

The concepts that make up the ontology model are not restricted to groupware systems: all interactive systems, including the single-user ones, should have an ontology model. But some aspects of it are more relevant to groupware systems than to single-user systems. For example, access and operation rights could be incorporated into a single-user system, although that is not usual. The same will be true for all aspects of the other two models described below: one can conceive a single-user system that might incorporate some of those concepts, but we believe those concepts are much more likely to be relevant for groupware than for single-user systems.

COORDINATION MODEL

The coordination model describes the activities that each participant may perform and how these activities are coordinated so that the group can accomplish its job. Let us first define the concept of activity, and the related concepts of s-action and t-action, and then discuss the other concepts related to the coordination models.

Some Definitions: Activity and Actions

The main concept of the coordination model is that of an activity. Other important concepts are role and actor. An activity is a potential set of operations (and the corresponding objects) that an actor playing a particular role can perform, with a defined goal.[1] In general, an actor may be a user, a computer system, or a group. The actor carrying out the activity is called the performer of the activity.

For example, in CA-ForComments there is the activity of creating the document, for which the available objects are the components of the document (e.g. lines, characters, and so on) and the available operations are

[1] The use of "potential" in the definition will become clear when we discuss the distinction between activity-level and object-level coordination below.

the usual edit operations on the document's components. The goal of the activity is to create a draft of the document, and the performer of this activity is the author. There are also review activities, one for each reviewer. The many review activities, which follow the create-document activity, can be performed in parallel or in sequence, depending on how the system is configured. The available objects for the review activities are the document, comments and revisions, and the operations available are: read (but not modify) the document components, create, modify and attach comments and revisions to the document components, and possibly read the other reviewer's comments, and create, modify, and attach dialogs to them. Finally there is the activity of incorporating the reviewer's comments into the document, performed by the author. The author may then choose to restart the review activities, or declare the document as finished.

A set of activities and the ordering among them make up a **procedure**. Thus, the activities above constitute the procedure of write/revise a document in that system. Usually groupware systems are designed for a single procedure (for example the review system above), but that is not necessarily so. For example a workflow system may embody an "order processing procedure" that incorporates the activities of credit check, billing, and shipping, and "payment of travel expenses procedure" among others.

Many customers who request to purchase goods from that company may be processed concurrently. Each order would represent an instance of the "order processing" procedure, and we call each of these instances an **endeavor**. As a procedure is made up of activities, an endeavor is made up of instances of activities, for example the credit check for customer Acme Inc. We call each instance of an activity (associated to a particular endeavor) a **task**.

We will call a task **inactive** if it has not yet started or if it has already been completed, and **active** otherwise. Thus active tasks are those activity instances that have already started, but which have not yet completed, and the operations and objects defined in each active task are still available to the actor that is performing it.

An activity is initiated by a start action or **s-action**, and completed by a termination action or **t-action** which is performed by one of the participants or by the system itself. Usually the performer of the activity is also responsible for one or all its s-actions and t-actions. In the review system example above, the reviewer may perform a t-action of submit comments to terminate his/her review activity.

An important aspect of a coordination model is the temporal precedence of the activities. Some activities can only start after others have been completed. Some activities may all be active at the same time, others not. For example in a particular workflow system one may require that all billing and shipping address information to be entered in a customer order form before the customer credit can be evaluated. But the system may allow that the entering of the billing address and the shipping address to be performed in any order, without any precedence between them.

The difference between an operation on an object and an action (a t-action or a s-action) is somewhat subtle but important. An activity is a sequence of operations on the objects available, performed by an actor. Thus, in a workflow system (for example FlowPath [2]), filling the fields in a form; in CAD systems, creating and modifying graphics objects; in a review system, creating and attaching comments, are all operations. But submitting the form for the next step of processing in the workflow system is a t-action. It significantly changes the stage of collaboration, for example, by giving to the next participant in the processing of the form access rights to it, and usually removing all rights of the first participant to that form. Furthermore, a s-action may be bound to a previous t-action.

The relation between actors and activities and t-actions may be a direct mapping, that is, the assignment of a particular person as responsible for an activity or a t-action, or can be mediated by a role. In this case, a role is assigned to activities and t-actions, and participants are assigned to roles. The assignment of activities and t-actions to roles is usually fixed and an essential part of the coordination model, while the participant-role mapping may be dynamic.

Activity-Level and Object-Level Coordination

In a groupware system, coordination or sequencing of events can happen in two levels: at the **activity-level** and at the **object-level**. At the activity level, the coordination model describes the sequencing of activities that make up a procedure. At the object level, the coordination model describes how the system deals with multiple participants' sequential or simultaneous access to the same set of objects. An example of activity-level coordination is the fact that a review system proposes/forces that the activity of writing the document must be completed before the activity of reviewing it can start.

Object-level coordination is concerned with how sequential or simultaneous access to the same objects are man-

aged. For example, how to deal with two participants' requests to modify the same object, at the same time. Thus object-level coordination sometimes utilizes solutions like locking, in which one of the participants would place a lock on an object and thus avoid simultaneous object modification.

Groupware tends to be especially concerned with shared access because this can enhance close inter-working of groups, and the synergy that makes groups productive and energized. Thus, a challenge at the object level of coordination is to allow more open access than traditional database solutions. Some groupware systems have pioneered non-locking shared write access, and interactive concurrency control [6].

Multiple and Single Endeavor

Some systems can only track/accompany/implement one endeavor at a time. For example, a group CAD system may only be able to deal with one design project at a time: the system can only hold one design that is being operated upon by a group of people. Of course one may start a second instance of the program on a second design, but the instances are not aware of each other.

Other groupware systems allow for multiple endeavors. For example, in the Coordinator, the procedure is the completion of a communication cycle as proposed by Winograd and Flores' theory of conversations for action [17]. The system can track many of those conversations, and in fact it was designed exactly to help with the difficult work of managing many conversations (endeavors). Similarly, a workflow system can also track or accompany the processing of many endeavors. In some way, both the Coordinator and workflow systems were designed not only to provide the capability of multiple endeavors but also to help each participant to manage them.

Stages and Inspection of Stages

In a coordination model each of the moments of the collaboration is called a **stage**. A stage is a global description of the instances of activities that are active. For example, one can say that the reviewing process is in the stage where reviewers A and B have both submitted their comments, and reviewer C is still entering his comments to the document.

Some systems are **single-stage**, that is the collaboration model does not provide any temporal sequencing of activities. Each of the participants are always engaged in a single task. This is common on systems with high levels of simultaneity, where the participants are all at the same time operating on the same set of objects. For example, a group CAD system, group drawing systems (for example CoDraft [9]) and group editors like GROVE [6] are all single stage. These systems exhibit a complex object-level coordination, at the expenses of a very simple, or in fact trivially simple, activity-level coordination.

Systems that are not single-stage and are multiple endeavor usually provide mechanisms to inspect the stage of each endeavor. There are many forms of inspection. The first one is the **participant-based inspection** in which the system will provide the information on all the active instances of activities a particular participant is responsible for, for all endeavors. In a review system, one could list all tasks that a particular participant has to perform, for example: review Smith's paper before May 3th, review Doe's paper before May 17th, and so on. Systems that provide this participant-based inspecting can also be seen as a managing tool for the users' tasks. They may provide tools to make this managing more effective: sorting tasks by urgency, allowing the participant to choose in which task he wants to engage next, and so on.

A second form of inspection is **endeavor-based inspection**, which will provide the global stage information about a particular endeavor. For example, a review system may allow the author of a document to check the stage of the endeavor of reviewing his paper. Such endeavor-based inspection would report, for instance, that Thompson has already submitted his comments, and that both Suzanne and Lee are still engaged in their respective reviewing tasks.

The Coordinator, for example, allows for both participant and endeavor-based inspection: one can list all requests that need reply (a participant-based inspection) or verify the stage of a particular communication (a endeavor-based inspection).

A third type of inspection is **total inspection** which combines both participant and endeavor-based inspections, and would display the stage of all endeavors. Possible examples of total inspection are workflow systems that provide such information to some privileged participant, like a manager.

The final form of inspection is **second-order inspection** which may provide statistical information on many endeavors like average time to complete this activity, average time for a specific participant to complete an activity, average time to complete the procedure, average number of tasks that a specific participant is engaged in at the same time, and so on. Again, the most

probable usefulness of such information is for privileged participants in workflow systems.

Finally, some systems may allow for the **modification of the stage**. That would be useful to deal with exceptions. For example, in a review system, if one of the reviewers calls the author and says that she cannot complete the review in time, it would be convenient to the author to alter the stage accordingly, in order to indicate to the system that that reviewer's task has terminated. In this case, the system will still wait for the other reviewers to submit their comments before the author could collate them, but it would not wait for this reviewer. The modification of the stage is an important aspect on the way the workflow system FlowPath deals with exceptions to pre-specified procedures.

Levels of Concurrency and Currency

The concepts of stages and multiple endeavors allow us to be more precise about the levels of simultaneity in a system. We will define the levels of simultaneity based on whether two or more tasks (instances of activities) can or cannot be active at the same time, and if at least two tasks can be active at the same time, whether they have or do not have access to the same set of objects.

The lowest level of simultaneity which we will call **sequential** will happen in single endeavor systems in which one activity follows the other. In such systems, only one instance of an activity is being executed at a time. An example of such a system is a single endeavor review system where the review activities are sequenced one after the other.

The second level of simultaneity, which we call **parallel** refers to instances of activities that can happen at the same time, but that belong to different endeavors. In this case, each task is dealing with a totally different sets of objects, and no interference between them can happen. Example of systems which allow for parallel activities are e-mail and coordination systems based on e-mail, like the Coordinator, Info Lens [12], etc. Each procedure in an e-mail system is strictly sequential: the activity of composing the message, and the activity of reading the message. But, in the system as a whole more than one instance of an activity may be active at the same time: Sousa may be composing a message, while Li-Chen is reading a different message. The activities are simultaneous but they do not share any object, since they refer to different endeavors. Similarly in the Coordinator, the activities to achieve the completion of a conversation are sequential but many endeavors can be going on at the same time, and thus many tasks may be active.

The third level of simultaneity, which we call **additive concurrent** refers to tasks within the same endeavor that can be active at the same time but that do not have modify rights to the same set of objects. That is, each participant can only add new objects, or place old objects into new relations (which, as discussed above, means creating a new instance of a relation object), and can only modify a subset of the objects, and this subset has no intersection with the corresponding subsets of any of the other participants. Since there is no simultaneous modification of any object, such systems do not need complex object-level coordination. An example of additive activities happens in an IBIS where each participant can only add objects and relate them to others, but cannot modify the existing objects, except the ones he owns.

Additive concurrent activities, (and systems that allow for them) also face the problem of **currency**, that is, how up-to-date are the objects being accessed by the participants. A participant may be accessing a set of objects that is no longer up to date. For example, a new object may have been created by a participant, or an already existing object has been modified, but that information has not yet propagated to the other participants in the system. **Notification time** is the time that any modification takes to be propagated to all participants. Furthermore, the updating mechanism can be **automatic**, that is, the process of keeping all versions of the objects in a current version is performed by the system. In a **manual** system, the participants are themselves responsible for initiating the updating process, usually performing a specific update action (for example, in most electronic mail systems, the user has to perform a get-new-mail action to receive the new mail that may have arrived).

The final level of simultaneity, which we call **fully concurrent** will happen in systems that allow for two or more simultaneous tasks to have modify rights to the same set of objects. These systems have to embody some form of object-level coordination to deal with concurrent modification of objects. [6] discuss some of the object-level coordination issues, under the name of "concurrency control." Fully concurrent systems also need a very high currency (usually they are automatic with very low notification time) since when a participant decides to modify an object she should be confident that she is accessing an up-to-date version of that object.

Furthermore one can extend the concept of concurrency of instances of activities to the system. A system that allows for fully concurrent tasks is a fully concurrent system. A system that does not allow for fully concurrent tasks but allows for additive concurrent tasks is an additive concurrent system. A system that does not allow

for concurrent tasks of any kind, but allows for parallel tasks is a parallel system. And finally a system that only allows for sequential tasks is a sequential system. [6] calls both additive concurrent and fully concurrent systems as real time systems.

USER-INTERFACE MODEL

Groupware reflects a change in emphasis from using the computer to solve problems to using the computer to facilitate human interaction. Users can best take advantage of this changed emphasis via systems with user-interfaces especially designed for groupware. We call these group user-interfaces. The issues that designers of group user-interfaces face are challenging and are significant extensions of the usual issues of interfaces for single-user systems. Thus our user interface model is highly concerned with representation of human-human interaction, and differs significantly from single user interface models. Our model has three components:

- views of information objects;
- views of participants;
- views of context.

Views of Objects and Local Operations

The user interface for a participant in a groupware session must be capable of presentation of the objects and operations embodied in the ontological model as previously defined in this paper. Since different participants may have different abilities (or different perspectives), the user interface model includes the concept of views of objects and the concept of local operations which are typically not present in single user models.

Besides the ontological model data, the user interface in a groupware system may have to deal with other "meta objects". Examples include telepointers and group windows [6]. For example, on systems that allow for inspection of the stage, the user-interface has to display this information and furthermore, if the system allows for alterations on the stage, it has to accept the operations that causes these alterations. This level of information corresponds to an interface to the aspects to the collaboration model of the groupware system that can be inspected and/or modified by the users. But, as we mentioned above, some systems may not allow for any form of stage inspection, and so this level of information may be absent in some systems.

Views of objects derives from the fact that different participants may want to have different views of the same objects of the system. For example, in a GDSS, a object may be semantically an array of numbers, but one participant may opt to view it as a line chart, another may prefer to see it as a pie graph, or as a table. All these different representations are views of the same object, and in principle it should be an issue related to the user interface. Furthermore, if one of the operations allowed for this generic array object is to alter the values of its cells, the operation has to be translated appropriately to each view of the object. In the chart view of the table, one could change the corresponding cell by clicking and dragging the point in the graph that correspond to the cell. In a table view, one could click at the appropriate cell and change its value using the keyboard.

Similarly, in an IBIS one participant may want to view the network of issues, positions and arguments as a graphic network of connected nodes. Another may want to view it as linearized text, indented appropriately to differentiate issues from positions from arguments.

The concept of local operations derives from the observation that group editing is a common and necessary operation in many groupware systems. There are decisions or options that must be built into each groupware system concerning granularity of edits and locality and when to transmit to others. If an edit operation is part of a real time synchronous interaction, then a WYSIWIS system may transmit the edit immediately to all participants. Alternatively, within an asynchronous system, edits may not be transmitted to other participants until a save operation is executed. For many existing systems, when entering data through the keyboard, the system considers all key presses as local operations until the return key is pressed. Thus composing a line is an atomic operation. Finally, it should be pointed out that in some systems, operations on meta objects, e.g. pointer movement, are permanently local operations which are never transmitted to other participants.

In a group editor system, for example, if the participant realizes that he mistyped the last character, he would press the backspace key and erase it. If we were to consider that within a real time interaction, it might be transmitted immediately. In GROVE and the Unix talk program, backspace key presses are object level operations, and are transmitted immediately. Clearly the level of granularity of operations depends upon the application and the group environment.

Views of Participants

Groupware is much more concerned with assisting people to people communication than most single-user sys-

tems. Providing some convenient means of knowing other participants, and what they are doing is an important aspect of our model. The identity of a participant is not directly related to the completion of the endeavor, but this information can be extremely helpful to the other participants in evaluating the situation of the group dynamics. For example, knowing that Smith is in a group long-distance discussion mediated by a IBIS, may lead the other participants to formulate their contributions in different ways than if Smith was absent. GROVE, for instance, provides this context information by displaying the pictures of the participants. Systems of video windows, video walls, virtual rooms, and virtual realities display the real time video images of participants which helps with the evaluation of everybody's attention and mood during the session. ClearBoard, for example, allows shared video drawing, while superimposing the image of the collaborating colleague [8]. This allows eye contact and gaze awareness, while still focusing on the work artifact.

In addition to displaying participants, it is possible to present, in an unobtrusive manner, relevant status, background, and preferences of participants. Benford discusses concepts of auras, nimbus, focus, and adapters [1], all of which are within the scope of the group user interface model. Group information such as the social network of who talks to whom can be presented, and the view of this can be tailored to the viewing participant. For example, if there is a relevant and significant shared previous experience between Smith and me, then I would like to be reminded, and associate this with Smith.

Other possible forms of participant context information are: how are the participants geographically distributed; what is the response time for long-distance connections; information from a database on relevant aspects of each participant; and so on. The information on the geographical distribution of participants may help long distance participants to realize that the subgroups that are in the same place may have developed other protocols of communication besides the one enforced by the collaboration model of the system. The response time information may help the division of labor among the group members so that the tasks of a long-distance participant should not depend on high currency. Finally, information about other participants may help a user to place the context of the other participants' contributions.

Views of Context

Another area of presentation that should be dealt with by a group user interface is all of the useful background material that we call context. The choice of what and how to present contextual information is a challenge, and that context may include items as diverse as the time of the next meeting, the current weather, and the presence of new mail messages from other participants.

We categorize contextual information as structural, social, or organizational. Structural context includes what and where data, such as the set of interactions in which I am currently participating, and temporal information such as what data has changed since I last accessed this hypertext web. Within a software engineering project, useful context may include languages and case tools used, status of various code and documentation files, and future milestones.

Social context includes items such as group norms, group metrics, and social history of the group. One proposed metaphor of shared virtual reality is that different projects would take place in dramatically different virtual rooms. Thus, our difficult design project would take place in a Tahitian hut, and our election processes always takes place in a London tea house; just the act of re-entering these contexts might trigger much useful contextual information in the heads of the participants. The research work on GroupAnalyzer [11] explored the efficacy of providing an electronic meeting barometer for groups in face to face interaction. This is an excellent example of the utility of graphical context presentation.

Organizational context can apply to small groups, large corporations, countries, or international organizations. It potentially includes formal reporting and responsibility structures of the group such as the organization chart. Also included are other items such as rules of the organization (procedures manuals, etc.) and inter-organizational data (competitive edge, mergers, etc.) In general, it must be understood that a meeting or any interaction is not an event in isolation, so these contextual clues provided by the user-interface can make the difference between a successful interaction versus a failure.

Finally, we note that participants are not context! Context connotes objects and conditions that are in the background. A primary function of groupware is support of communication and collaboration among participants, so participants are in the foreground.

Issues for Concurrent Systems

Concurrent systems, both additive and fully concurrent, present some special concerns to the user interface, related to the problem of currency. The first concept is that of **notification level**, that is, how much of

the modifications to the objects done by the other participants should be notified to a user. Too much of such notifications may be distracting to the user, too few will lower the user's currency below an acceptable limit. Also in this issue, different notifications may be displayed in different ways, to attract or possibly not to attract the user's attention. Thus, changes made on objects in which the user is currently interested should attract his attention, whereas changes made on objects of less interest should not. [6] discusses some of the particular issues of user-interfaces in concurrent systems.

Also in concurrent systems, it may not be wise to have total tailorability for the user-interface. If two users have very different user-interfaces, and different views of the same objects, they cannot share their experiences outside the context of the groupware as has been noted in the literature. Thus, if two people are working in a concurrent system and one asks for some help on using some features of the user interface to the other, if the views and interfaces are very different, then the second one may not be able to help the first. How much of such tailorability of interface and views is an issue that must be carefully addressed.

SUMMARY OF THE CONCEPTUAL MODEL AND CONCLUSIONS

The conceptual model of groupware is a view of groupware based on three aspects: the description of the "things" and operations on these "things" that the system provides to the user, which we called the ontological model; the description of how the activities of each participant should be temporally organized and coordinated, which we called the coordination model; and how are the users to interact with the system and each other, which we called the user-interface model.

Existing systems seem to present some regularities on all three models, especially in relation to the coordination model. For example, the available fully concurrent systems are all single stage, that is they have a very complex object-level coordination and a trivially simple activity level coordination. On the other hand, systems with more elaborate coordination models, like review systems or workflow, tend to not be fully concurrent. We expect that in the future of groupware, these simple regularities will yield to more diversity.

It is also important to notice that some systems seem not to fit well into this three-sided description of groupware. Info Lens and Lotus Notes [10] are some of them. We believe that such systems should be classified as **meta-groupware** systems, and not groupware system themselves. For example, Info Lens does not provide objects and operations (an ontological model) but it provides the possibility of defining classes of objects or message-types. A user may write rules that define what are the classes of message-types (the ontological model) and rules that define what should be done upon receiving each message-type (the coordination model). Once the rules are written, that instance of Info Lens becomes groupware itself, with its own ontological and coordination models. Similarly, Lotus Notes by itself does not define either objects or activities, but its scripting language allows one to create an instance of a groupware systems, with both an ontological and a coordination model.

We call these systems meta-groupware because they could be seen as applications whose objects are groupware systems. Both Info Lens and Lotus Notes are *groupware builders*, that is they create instances of groupware systems, but in general building groupware systems is not the only thing a meta-groupware could do: one could conceive meta-groupware that not only builds but also analyzes a groupware system, or one that would help the users to dynamically change for example the coordination model of a groupware systems while it is in use [7].

The work presented in this paper is only the first step towards of what we call a conceptual model of groupware. This paper has presented some intuitions behind the three-models approach to classifying groupware. This tri-model approach has not yet been formalized but even at this intuitive level this model allows one to define important concepts and distinctions. We believe that this is the main contribution of this paper. Even at the intuitive level, concepts like inspection of stages, multiple endeavor systems, intended and operation semantics of objects, and so on, allow one to classify and describe the main aspects of a groupware system. Maybe even more important, it allows a designer of a groupware system to express and explore his/her intuitions on what are the important aspects of the system to be build.

This work will be extended in many directions: development of some formalization language for the ontological and coordination models; expansion and elaboration of the user interface model which seems to be crucial within future tightly coupled groupware systems; further exploration of the concepts related to meta-groupware systems: there seems to be the need for concepts like meta-ontological model, meta-coordination model, and meta-interface model in order to describe the set of possible groupware systems a groupware builder can construct.

REFERENCES

1. Benford, S.A. Spatial Model of Interaction in Large Virtual Environments. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, (1993)
2. Bull Corporation *FlowPath Functional Specification.* (Sept. 1992) Paris, France, .
3. Broderbund Software *CA-ForComments 2.5 PC User Guide.* (1991) San Rafael, California.
4. Conklin, J. and Begeman, M. gIBIS: An hypertext tool for exploring policy discussion. In *Proceedings of the Second Conference on Computer Supported Cooperative Work*, (1988)
5. Dourish, P. Culture and Control in a Media Space. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, (1993).
6. Ellis, C.A. and Gibbs, S.J. and Rein, G.L. Groupware: some issues and experiences. *Communications of the ACM* 34 (1991)
7. Ellis, C.A. and Keddara, K. Dynamic Change within Workflow Systems Technical report CU-CS-667-93, Department of Computer Science, University of Colorado, (Aug 1993).
8. Ishii, H. and Kobayashi, M. Integration of Interpersonal Space and Shared Workspace. In *ACM Transactions on Information Systems*, 11,4. (October 1993).
9. Kirsche, T. *et all.* Communication Support for Cooperative Work. *Computer Communications*, 16(9), 594-602.
10. Lotus Corporation *Lotus Notes Application Developer's Reference (Release 3)* (1993)
11. Losada, M. and Markovitch, S. GroupAnalyzer: A System for Dynamic Analysis of Group Interaction. In *Proceedings of the 23 Annual Hawaii Internatonal Conference on System Sciences.*
12. Malone, T.W., Grant, K.R., Turbak, F.A. The Information Lens: An intelligent system for information sharing in organizations. In *Proceedings of the ACM SIGCHI Human factors in Computing Systems.* (1986)
13. Neuwirt, C.M. and Kaufer, D.S. and Chandhok, R. and Morris, J.H. Issues in the Design of Computer Support for Co-Authoring and Commenting. In *Proceedings of the Third Conference on Computer Supported Cooperative Work*, (1990)
14. Opper, S. A groupware toolbox. *Byte* (Dec. 1988)
15. Rein, G.L. and Ellis, C.A. rIBIS: A real-time group hypertext system. *Int. J. Man-Machine Studies* 34,3 (1991)
16. Schmidt, K. and Bannon, L. Taking CSCW Seriously. In *Computer Supported Cooperative Work Journal*, 1,1 (1992).
17. Winograd, T. and Flores, F. *Understanding computers and cognition.* Ablex, NJ, 1986