Collaboration over Wired and Wireless Networks

Ivan Marsic

Abstract--Digital networking has become pervasive, bringing advantages of distributed computing and collaboration that are time and space independent. But, users typically employ devices with widely disparate capabilities. Design of networked systems that can accommodate such disparities, while achieving maximum utility, is therefore a key research issue. Real-time application sharing in heterogeneous environments requires solving many software-architectural and networking quality-of-service problems. The research efforts towards this goal are very few and very recent. This report presents our framework for developing software applications that deal with heterogeneity and our experiences with its implementation and deployment in an academic environment. We present a software architecture and evaluate human performance for collaboration in heterogeneous virtual worlds.

Index Terms--Collaboration systems and technologies, groupware, wireless and mobile computing.

I. INTRODUCTION

The recent development of the Internet with wireless links and portable terminals presents users with significant disparities at several levels: network capabilities (bandwidth, delay, packet loss), interaction modalities (interface devices, displays), computing power and accessible storage. Disparities can exist at both network and client-station levels. Networking might be broadband wired service in one instance, and mere voice-bandwidth wireless in another. Client equipment might range from high-end workstations with multimodal interfaces and elaborate graphics, to cell phones, beepers, or palmtops. In this paper all of these features are addressed.

The heterogeneous Web with wireless and wired links is seen as the most important future development in information technology. We believe that interactive applications will play an important role. Collaboration of geographically dispersed knowledge workers is one of the key applications that will provide value to the infrastructure. A simple collaboration example in wireless networks is two-way real-time text chatting to replace current two-way pagers. As small portable devices advance in visualization capabilities, more complex workspace sharing is just a step ahead. An example of such visualization is the work on miniGL [1], an OpenGL-compatible 3D graphics library for PalmPilot handheld computers.

Heterogeneity naturally occurs in practical tasks. Applications for synchronous collaboration are emerging in many fields, e.g., business, healthcare, military services and transportation. Fig. 1 illustrates several scenarios in which ubiquitous collaboration is key. The classic example is the provision of just-in-time assistance between a deskbound expert and a mobile fieldworker using a portable device. The mobile worker may work with blueprints and the expert works with a 3D CAD model to repair a vehicle or the plumbing in a building. In a firefighting situation, an office worker walking through a virtual representation of a building may assist firemen who work with a blueprint of the building.

Sharing heterogeneous applications raises issues at both the machine level and the human level. At the machine level, we have to deal with synchronization of heterogeneous application states—the states need to be mapped between the disparate domains and maintained in synchrony in real time. At the user level, the basic premise of real-time collaboration—that efficient reference to common objects depends on a common view of the work at hand—is no longer maintained since the users may be presented with significantly different views.

Our work on the DISCIPLE [2] and Manifold frameworks supports the development of collaborative applications for these heterogeneous environments. Using eXtensible Markup Language (XML) for the communication medium provides a solution for the heterogeneity. Collaborators share the same data or a subset of the data, represented in XML, but view very different displays depending on their computing capabilities. Mobile users, for example, may view a more limited presentation than their collaborating colleague in the office. The Manifold framework transforms information to match the client's local capabilities and resources, yet maintains important content information. We used DISCIPLE and Manifold to implement a 3D to 2D collaborating environment and measured team performance characteristics in a simple object placement task.

Multimedia applications typically show considerable computing complexity and require stable network quality-of-service (QoS) in terms of throughput, bounded delay, and small jitter. Wireless connectivity and limited resources make the provision of QoS for multimedia applications on mobile devices a challenging problem. There are essentially two ways to provide QoS for network applications. One is

Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854-8058 USA.

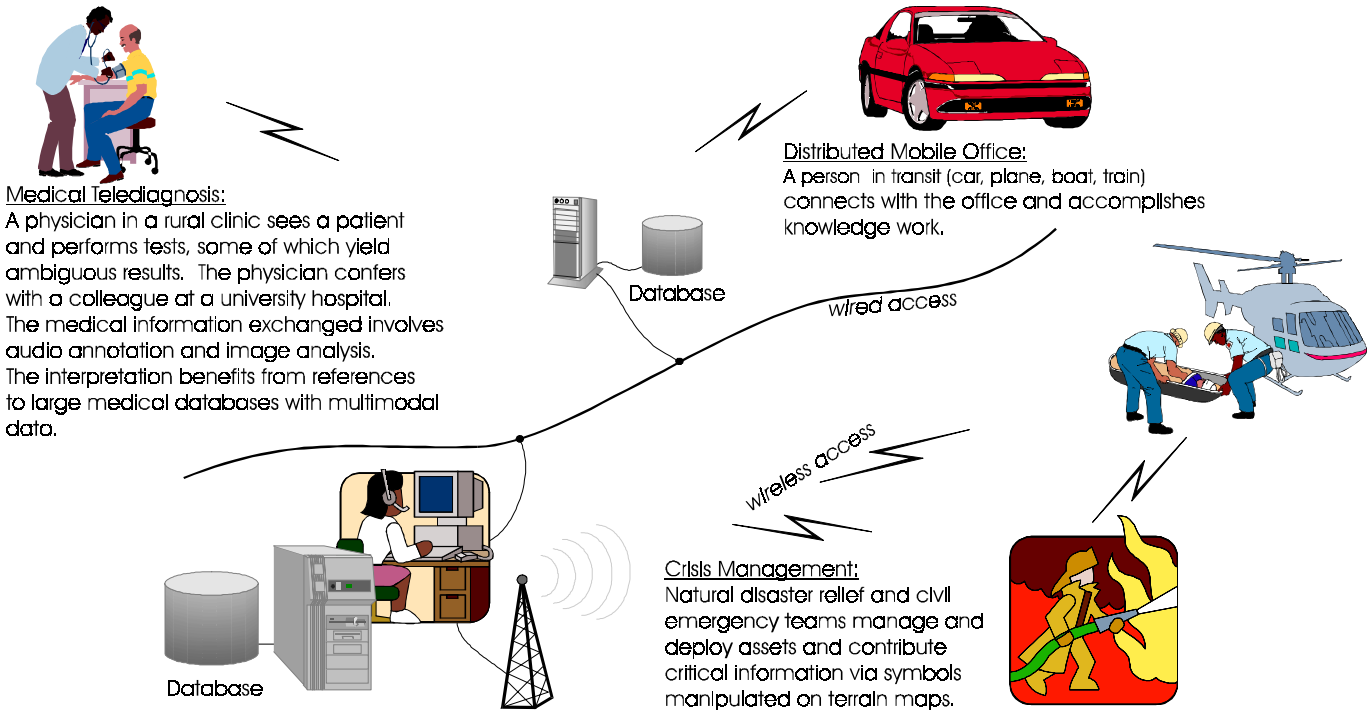

Fig. 1. Examples of environments and collaborative tasks illustrate the envisioned benefits and provide testbeds for quantitative evaluation of ubiquitous collaboration. All illustrated activities are supported by shared electronic workspaces.

by resource reservation from the network's viewpoint; the other is by behavior adaptation from the application's viewpoint. We adopt the latter approach. Knowledge of the characteristics of the network environment is the basis of adaptation. An important objective of our research is development of an architecture for wireless awareness which will enhance the performance of collaborative multimedia applications in heterogeneous networks.

The paper is structured as follows. Section II reviews related work in this area. Next, Section III presents our software architecture for collaboration in heterogeneous environments. Then we present the role of XML in managing heterogeneity in Section IV. Section V describes intelligent agents for network and computing environment awareness. Section VI presents a short study that investigates the feasibility of collaboration in heterogeneous environments from the human factors perspective. Finally, we conclude the paper and propose future work.

II. Background and Related Work

Many collaborative systems and toolkits address the issues of sharing across dissimilar terminals. For example, Garfinkel *et al*. [3] discuss at length the adaptation of the shared display to various display devices using SharedX. If necessary, SharedX degrades the quality of the displayed images to match the capabilities of the display hardware. However, the authors assume that users have the same replicated application and are just applying device-specific rendering. Rendezvous [4], GroupKit [5] and several groupware toolkits thereafter use model-view separation so that developers can create models and drive different views. However, no implementation is attempted, and in some cases (e.g., [4]) the situation is greatly simplified by using centralized groupware architecture.

An important aspect of heterogeneity is interoperability between the groupware systems. Dewan and Sharma [6] address this issue. Another approach [7] is grounded on the premise that almost any information can be encoded as 2D pictures, and that humans readily understand such pictures. The focus is thus on a common pictorial surface that contains a rendering of graphical objects. Sharing between collaborators is implemented at the level of the pictorial surface (i.e., view), with PostScript as the communication standard. As the authors point out, editing the surface is cumbersome since application logic is lost in the rendering process. Our work is complementary to these efforts insofar as we focus on an architecture for developing new applications for heterogeneous environments, rather than interoperating the existing applications.

Using XML for data exchange is key to our architecture for heterogeneous groupware. XML from its inception was intended to separate data representation from visualization. To accomplish this, data are described in the XML document whereas the rendering is determined by an XSL (eXtensible Style-sheet Language) document. Several efforts are underway to realize XML's potential for information exchange on heterogeneous devices. The Wireless Application Protocol (WAP) Forum has standardized Wireless Markup Language (WML), an XML language optimized for specifying presentation and user interaction on limited capability terminals such as mobile

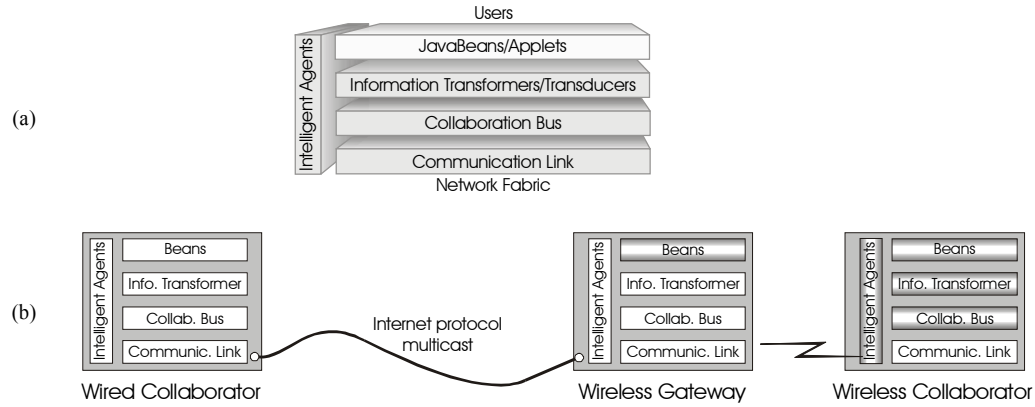

Fig. 2. Architecture of the DISCIPLE collaboration framework. (a) Components of the framework. JavaBeans/Applets are not part of the framework; they are supplied by the application developer rather than by the framework developer. (b) Distribution of components in a heterogeneous environment. Wireless clients comprise some or all of the shaded components depending on the capabilities of the access device.

phones, pagers, two-way radios, and smartphones [8]. Several companies, e.g., Nokia, already offer WAP-compliant devices, but this architecture currently does not utilize XML/XSL separation. WearLogic developed an XML data model and browser for a credit-card-size personal information organizer [9]. It implements simple data rendering, but not arbitrary applications. To our best knowledge, there is currently no other research effort that uses XML/XSL separation for dynamic run-time adaptation of application models and visualizations to diverse devices.

III. System Architecture

The DISCIPLE system is a mixture of client/server and peer-to-peer architecture. It is based on a *replicated architecture* for groupware [10]. Each user runs a copy of the collaboration client, and each client contains a local copy of the applications (Java components) that are the foci of the collaboration. The architecture of the DISCIPLE framework is shown in Fig. 2. As a first approximation, all wired clients are assumed to belong to one class and all wireless clients belong to another class (Fig. 2b).

The Intelligent Agents Plane is responsible for ensuring that its client is an effective participant in a heterogeneous collaboration session. It uses a knowledge base to track and translate the client's interests, computing capabilities and quality-of-service (QoS) requirements into a *client profile* that defines the type and level of information abstraction needed. The profile is then used to visualize the incoming data or to modify outgoing events using the services of the information transformer module.

The Information Transformer/Transducer Module maintains a suite of media-specific information abstraction algorithms. Information abstraction aims at intelligently reducing information content while maintaining semantics. Information-transforming processes assist the bandwidth- and display-disadvantaged mobile wireless clients. Examples of information abstraction include image-to-text, image-to-speech, text-to-speech, and speech-to-text conversions. Such a translation is critical for enabling collaboration across heterogeneous clients and interconnects with large discrepancy in computing capabilities.

The central part of DISCIPLE is conceptualized as the Collaboration Bus [11]. It spans network fabrics and provides a virtual interconnect for geographically distributed clients. The bus achieves synchronous collaboration through real-time event exchange, dynamic joining and leaving, concurrency control and crash recovery. The collaboration bus comprises a set of named communication channels, which the peers can subscribe to and publish information. In order to make the user aware of other users actions, the DISCIPLE GUI provides several types of group awareness widgets to all the imported beans [12]. For example, telepointers are widgets that allow a given user to track remote users' cursors. In addition, users can exchange messages, post small notes, and annotate regions of the shared application window.

A. Sharing Java Beans

DISCIPLE is an *application framework* [13], i.e., a semi-complete application that can be customized to produce custom applications. The completion and customization is performed by end users (conference participants) who at runtime select and import task-specific Java Beans [14]. The DISCIPLE workspace is a *shared container* where Beans can be loaded very much like Java Applets downloaded to a Web browser, with the addition of group sharing. Different Beans represent different application plug-ins for a generalized application-sharing infrastructure provided by DISCIPLE. Collaborators select which "plug-in" to load based on the task at hand and import the selected Bean by drag-and-drop manipulation into the workspace.

The imported Bean becomes a part of a multi-user application and all conferees can interact with it. Objects in

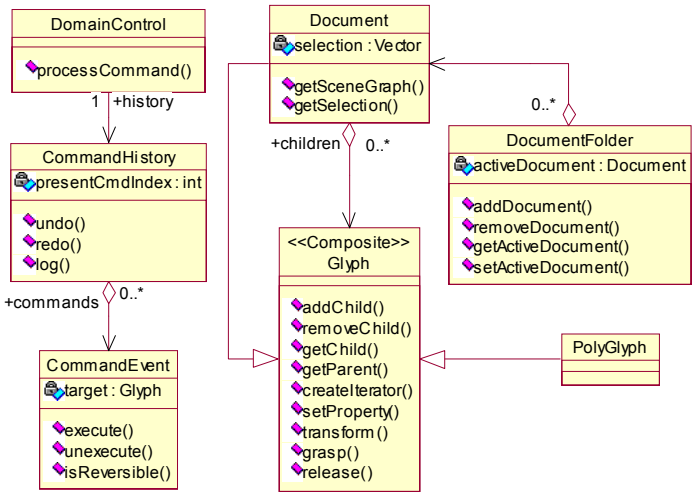

Fig. 3. Domain class diagram of a generalized editor represented in Unified Modeling language (UML) notation.

the Bean are not aware of distributed services or sharing. They interact with DISCIPLE through the Beans event model as with any other event source/listener. DISCIPLE provides for distribution.

The application framework approach has advantages over the commonly used toolkit approaches in that with toolkit approaches the application designer makes decisions about the application functionality whereas in our approach the end user makes decisions. We consider the latter better because it is closer to the reality of usage and the real needs of the task at hand.

B. Manifold Beans

DISCIPLE provides several features that handle heterogeneity primarily at the communication/networking level [15], but not at the application logic and interaction levels. For this, the application Bean, itself, must be properly "architectured." We present a class of Beans called **Manifold Beans**, which are designed to deal with heterogeneity at these levels.

The main characteristic of the Manifold beans is the multi-tier architecture. The common three-tier architecture comprises the vertical tiers of presentation, application or domain logic, and storage. The Manifold's *presentation* tier is virtually free of the application logic and deals with visualizing the domain data and accepting the user inputs. The *domain* tier deals with the semantics of tasks and rules as well as abstract data representation. The third tier in our case comprises the collaboration functionality, which is mainly provided by the DISCIPLE framework, but a part of it resides in the Manifold bean as discussed below.

The key concept is a Glyph, which represents all objects that have a geometry specification and may be drawn. The name "glyph" is borrowed from typography to connote simple, lightweight objects with an instance-specific appearance [16]. Fig. 3 shows a simplified class diagram. All Glyphs in a document form the scene graph, itself a Glyph, which has a tree data structure. The scene graph is populated with different vertices (Glyphs) in specific applications that extend the Manifold framework. Glyphs are divided into two groups. Leaf Glyphs represent individual graphic elements, such as images, geometric figures, text or formulas in spreadsheet cells. PolyGlyphs are containers for collections of Glyphs. They correspond to branch nodes and can have children. Example PolyGlyphs are group figures, paragraphs, maps or calendars. PolyGlyphs have all the functionality of the Glyphs and the additional property that they can contain Glyphs or other PolyGlyphs.

A Glyph is essentially a container for a list of <*property*, *value*> pairs, very much like XML elements. The operations on the Glyph tree are: create Glyph (Op_1), delete Glyph (Op_2), and set Glyph properties (Op_3). The corresponding Glyph methods are `addChild()`, `removeChild()`, and `setProperty()`. Properties include color, dimensions, constraints on glyph manipulation, etc.

Glyphs are sources of the following types of events which are fired in response to the operations: AppearanceEvent for add/remove operations, PropertyChangeEvent, and TransformEvent. The interested parties register as event listeners for some or all types of the events via the Java Beans delegation event model.

DomainControl is the system controller for the domain bean that invokes the system operations. This is the only portal into the domain bean. The only way to cause a state change in the bean is to invoke the `processCommand()` method. Even the local presentation (view) objects interact with the domain objects through this portal only.

The CommandEvent class implements the Command pattern [16] and has the responsibility of keeping track of the argument values to invoke operations on Glyph and Document objects so the operations can be undone/redone. We name this class CommandEvent instead of Command to emphasize the Java event distribution mechanism.

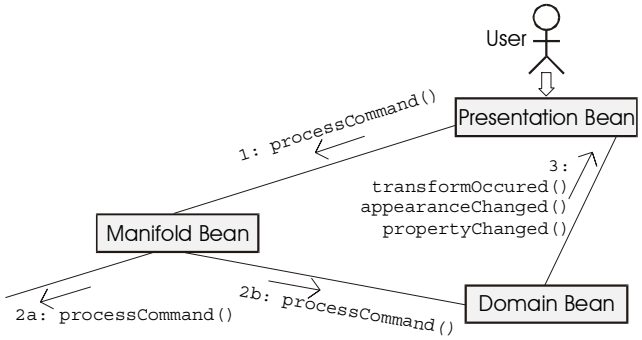

Fig. 4. Event exchange and interception in a Manifold bean. Commands generated by the local user that affect the domain bean. Action 2a passes the command to the collaboration bus which broadcasts it to the remote peers.

Commands create/delete Glyphs or correspond to Glyph methods. In addition, we have commands to open or save a document and document-view-related commands. CommandEvents include the user ID attribute so it is possible to identify the originator of the command. This is useful for group undo/redo as well as for group activity awareness and for controlling access rights.

The domain and the presentation beans need to be glued together since the execution environments assume single beans. The third bean, the Manifold Bean, interacts with the collaboration bus to send and receive collaboration events. This interaction is through the Java Beans' event mechanism: DISCIPLE subscribes for the bean's events and replicates them to the collaborating peers. Fig. 4 shows the interactions between the beans. A local command is dispatched to the local domain bean and the collaboration bus either simultaneously or first to the bus and then locally after the command is reflected back. Which strategy is used depends on the employed concurrency control algorithm, optimistic or pessimistic.

IV. XML AND MANIFOLD BEANS

So far we have presented an architecture which can be used to build heterogeneous groupware. However, once built, the beans do not allow for dynamic customization and thus, adaptation to the current context. Since Java supports dynamic loading of individual classes, we could exploit this feature and consider the above classes as components in a repository, which can be selected at run-time so to customize the application's document, the way it is visualized, and the way the entire application is generated. We have chosen XML as the language for such purpose.

The XML programmer creates XML and XSL files based on, respectively, the sets of available Glyphs and GlyphViews as well as their characteristics. Manifold Glyphs correspond to elements in the source tree and GlyphViews correspond to elements in the result tree. Attributes in the result tree are mapped to the Glyph properties. For example, if the *result node* corresponding to a Glyph has an attribute COLOR with a value of BLUE, the Glyph's property "color" is initialized in blue. However, the correspondence between the elements and Glyphs is not one-to-one. Not all XML elements are Glyphs. For example, some elements describe Glyph properties. A 3D transformation could be represented as:

```
<TRANSFORMATION>
… 4x4 matrix defining the 3D transformation …
</TRANSFORMATION>
```

This element could even have sub-elements if the transformation is represented via the axis-angle, scale and translation parameters, each tagged individually.

The result tree is used to instantiate the domain scene graph. A key benefit of implementing presentation and domain as distinct beans rather than the whole package as a single bean is being able to mix and match different

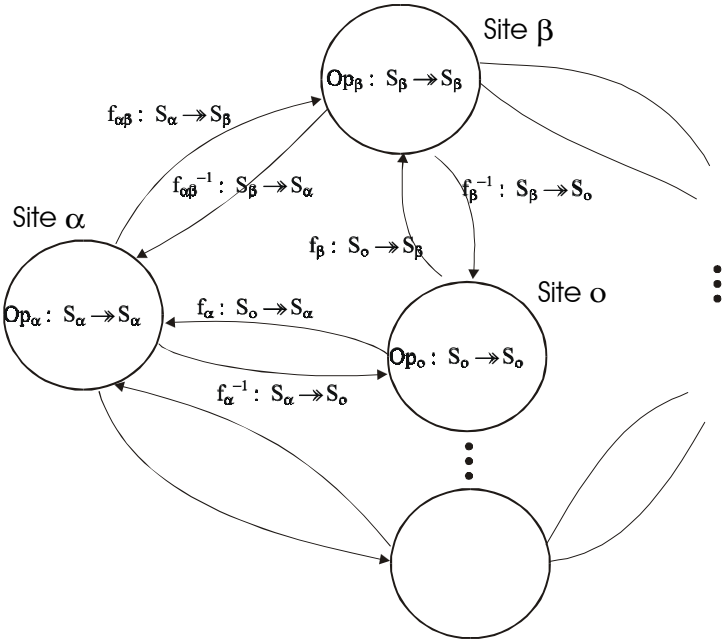

Fig. 5. Operations and mappings between the domain scene graphs (S) in a heterogeneous collaborative system. A central site o could maintain a common model to simplify the mappings to/from different sites.

combinations. We can have a set of more or less complex beans for each. Different domain beans can implement complex behaviors and the presentation beans can implement visualizations with varying realism. Depending on the context, different domain/presentation pairs can be downloaded on demand. In addition, each bean in the pair may run on different machines. The presentation bean may run on a hand-held device, and the domain bean would run on a PC embedded in the environment (base station).

In heterogeneous applications, the scene graphs are populated with different Glyphs having different properties. The Glyph operations do not directly apply to remote Glyphs. We need to transform the operations to apply them on remote sites (Fig. 5). In the worst case, when all n sites have different domains, there are $n*(n–1)$ mappings. The agency that does the mappings can be a central server or, in an entirely distributed case, each client can perform the required mappings on its own. This latter approach is not likely to be practical for lightweight terminals, so proxy agencies are needed to perform the mappings.

For instance, in the example applications given in Section VI, the event communication is implemented as follows. In order not to have the lower-end machines do the mapping, multiple communication channels are used. The low-end beans (called Flatscape) only communicate on their dedicated collaboration bus channel, while the high-end beans (called cWorld) communicate on both the Flatscape channel and the cWorld channel. The mapping and translation between the two domains is therefore only performed on higher-end machines (running cWorld).

V. Intelligent Agents and Network Awareness

To deploy the data-centric scheme, the system must build the environment profiles. This is the task for intelligent agents, Fig. 2a. Agents span all layers of the architecture since they need information from all layers to make intelligent decisions.

Agents sense the networking environment and determine what transform to apply to the information exchanged between the conferees to match diverse data representation domains and computing and network capabilities. Agents also dynamically determine the composition of the Manifold bean based on the capabilities of the client device and other components of the profile.

An example task for intelligent agents is determining the characteristics of the communication links. This information may include available throughput or confidence of the existence of wireless links. It is then used by the DISCIPLE framework to adjust to the computing environment in such a way to automatically apply different information transformations and XSL documents for data visualization. Here we give a brief overview of our architecture for wireless awareness that enhances the performance of network applications in heterogeneous communication environments, where both wired and wireless links exist. Further details are available in [15].

A. Wireless Awareness Architecture

We investigate the existence and characteristics of wireless links from both the end-host's and network's viewpoints. We currently employ three methods for wireless-awareness: (*i*) an end-host method by accessing device drivers of network interface cards to obtain the channel parameters of local wireless links, (*ii*) a network method by instrumenting round-trip time (RTT) values of the communication entities to detect the existence of remote/local wireless links, and (*iii*) a network method by processing the inter-packet time to estimate the available bandwidth of remote/local wireless links, including the asymmetric links. Note that the latter two methods can be applied to gain the awareness of both local and remote wireless links.

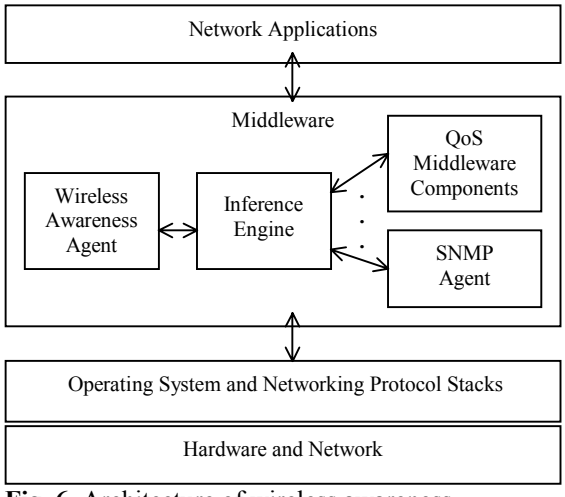

Fig. 6. Architecture of wireless awareness.

The wireless-awareness architecture is shown in Fig. 6. The component that deals with wireless awareness is an intelligent agent. The wireless-awareness agent can communicate with other QoS-middleware components and QoS-related agents, such as SNMP (Simple Network Management Protocol) agent. The agent's inference engine resolves the ambiguities and uncertainties in the gathered data.

The wireless awareness agent may implement different awareness techniques or delegate the measurements to other agents, such as the SNMP agent. In the minimum configuration, it contains only the inference engine. This makes the architecture lightweight and deployable to small devices.

The wireless awareness agent detects, measures, integrates, and manages the wireless-link-related parameters. The agent detects the existence of the wireless links and measures the available wireless bandwidth and other parameters of the wireless channel. DISCIPLE deploys wireless awareness by querying the wireless awareness agent according to the application bean's QoS requirements or by subscribing to the information broadcast by the agent. The generic procedure for interaction between an application and the agent is illustrated in Fig. 7. This procedure can be used in both the startup phase and the runtime phase of network applications. In the startup phase it is used in the proactive mode, while in the runtime phase it may be used in both proactive and reactive modes.

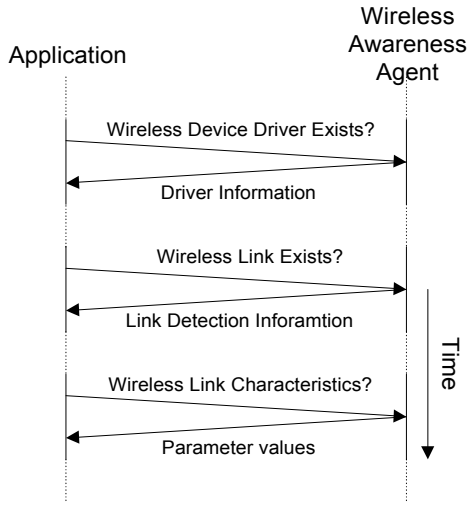

Fig. 7. Communication procedure between network applications and wireless awareness agent.

B. Example: Available Bandwidth Awareness by Inter-Packet Time Processing

An example of characterizing the communication channel is measuring the available bandwidth, which is determined by the bottleneck link bandwidth. The essential idea is using inter-packet time to estimate the characteristics of the bottleneck link. If two packets travel together so that they are queued as a pair at the bottleneck link with no packet intervening between them, then their inter-packet spacing is proportional to the processing time required for the bottleneck link to transmit the second packet of the pair (Figure 6). We modified the technique presented in [17] to apply it to asymmetric wireless links as well.

C. Heterogeneity Management

We could implement one general domain bean that contains many different Glyphs and Behaviors. Also, a corresponding presentation bean could render any object, whether two-dimensional or three-dimensional. The extreme solution would be to make no distinction between the beans. However, these solutions are very inflexible and particularly inappropriate for thin clients which cannot handle large programs. For example, if a Glyph represents a vehicle that can be set into motion, the Glyph's behavior should be pre-programmed to act as a mobile vehicle and cannot be altered. For every new behavior, we have to alter the domain or presentation packages and introduce new Glyphs or Behaviors.

A key benefit of implementing presentation and domain as distinct beans rather than the whole package as a single bean is in being able to mix and match different combinations. We can have a set of more or less complex beans for each. Different domain beans can implement complex Behaviors and the presentation beans can implement visualizations with varying realism. Depending on the context, different domain/presentation pairs can be downloaded on demand. In addition, each bean in the pair may run on different machines. The presentation bean may run on a hand-held device, and the domain bean would run on a base station (wireless gateway in Fig. 2b).

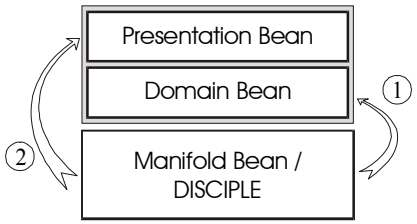

Fig. 8. Bean loading mechanism for heterogeneous clients. ① The control module (Manifold Bean equipped with the information from DISCIPLE) downloads the domain bean based on the computing/network environment characteristics. ② The control module then downloads the XSL document, parses it and downloads the presentation bean.

In this solution the user selects which overall bean to load for the task at hand, and which domain/presentation beans should be loaded as its components. Having the user manually select domain/presentation beans may be overwhelming. Instead, this process can be automated. The "glue" bean can assume the role of automatic bean selection since it loads and interconnects the other two beans. Ideally, we should be able to dynamically select the bean composition based on the capabilities of the client device. The intelligent agent senses the computing and network environment and loads the other two beans accordingly. It also selects the XML and XSL documents to instantiate the beans. The process is illustrated in Fig.8. The user's current task determines the corresponding Manifold bean. The DISCIPLE framework senses the environment capabilities, such as display size or network bandwidth and passes this information onto the Manifold bean [15]. The Manifold bean then loads the domain/presentation beans pair and the client is ready for the collaborative session.

An even more flexible and advanced solution is available by using the Bean Markup Language (BML) [18] to assemble the beans at run time. Instead of directly loading the pre-packaged domain/presentation beans, the Manifold bean may load two BML documents. The BML documents contain scripts for dynamic assembling of the beans from a (remote) repository of Java classes. We are presently working on implementing this solution.

Even though Manifold is relatively lightweight, it is still a problem to run DISCIPLE and a Manifold bean on a PDA such as a Palm Pilot, mainly because of memory limitations. Our solution to this problem is to run only the presentation tier on the client PDA, while DISCIPLE and the other two tiers of the Manifold bean are running on a wireless gateway. This is illustrated in Fig.9. Plain TCP/IP connections are used for communication between the presentation bean on one end and the collaboration and domain beans on the other end (see Fig. 4).

VI. Test Applications and Experiments

Two issues are pertinent for the presented framework:

- How well it generalizes to arbitrary applications?
- Can users collaborate equitably?

The architecture of the Manifold framework is lightweight, scalable, and extensible. Dewan [10] argues that any collaborative application can be seen as a generalized editor

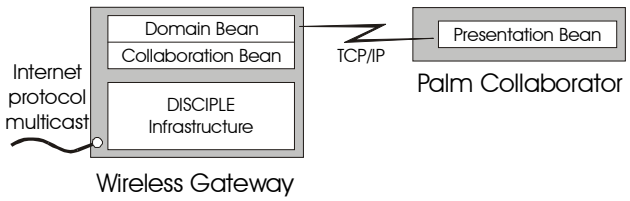

Fig. 9. The distribution of Manifold Beans for a lightweight PalmPilot-based client.

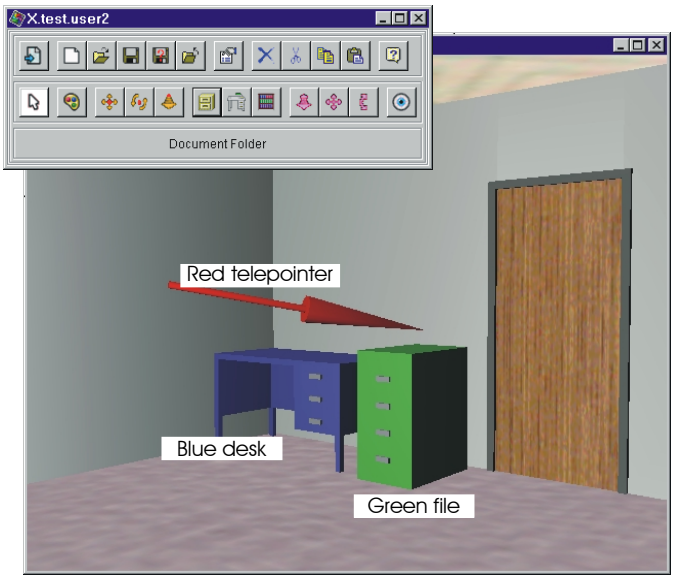

Fig.10. A sample CVE built using cWorld. The figure also shows a three-dimensional telepointer.

of semantic objects defined by it. A user interacts with the application by editing a rendering of these objects using text/graphics/multimedia editing commands. In addition to passive editing, in a generalized editor, the changes may trigger computations or behaviors in the objects. In that sense, Manifold can be used to build arbitrary groupware applications that deal with structured documents.

The set of applications implemented and tested to date includes text-based chat, whiteboard, 3D collaborative virtual environment (CVE), collaborative situation map, speech signal acquisition and processing, image analysis tools, and a medical image-guided diagnosis system for the diagnosis of leukemia [19]. The example applications given below use Manifold for graphics editors with 2D vs. 3D representations of objects.

A. Example Applications

The cWorld Java Bean (Fig. 10) enables synchronous, multi-user building of collaborative virtual environments (CVEs). It uses Java3D. CWorld does not require special hardware and can be operated using the keyboard and a mouse although it also supports the use of the Magellan SPACE Mouse. This device provides the six-degrees of freedom movement used in navigating 3D spaces.

The primitive operations on the Glyph tree in cWorld are:
Op_1: creates simple geometric figures (box, cylinder, cone, sphere), lights (directional, point, spotlight), and furniture objects (desk, cabinet, bookcase), which are PolyGlyph objects. The room object is a PolyGlyph as is the telepointer (cylinder + cone).
Op_2: deletes any of the above listed Glyphs
Op_3: sets the Glyph properties, such as color, texture, position, dimensions, and manipulation constraints (e.g., furniture objects are constrained from 'flying').

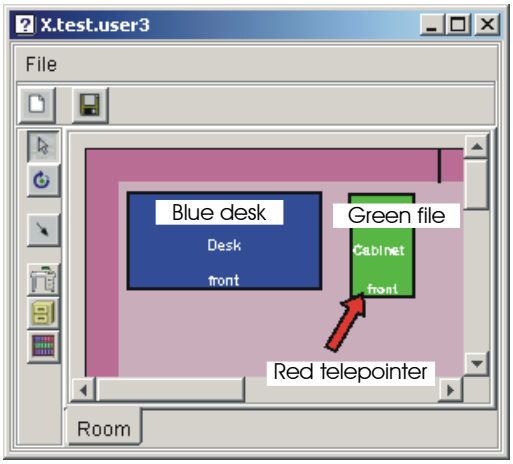

Fig.11. A room floor plan as seen using Flatscape, corresponding to the room shown in Fig.10. The figure shows also a two-dimensional telepointer.

The lights have additional properties, such as direction, attenuation, spread angle, etc. The user can also apply 3D affine transformations to Glyphs.

User commands result in one or more primitive operations. For example, moving a figure to background/foreground or grouping/ungrouping figures requires several primitive operations.

Each user has a unique 3D telepointer, shown in Fig. 10. These devices function as primitive avatars and appear at the discretion of the user. The telepointer is drawn at the position and orientation of the user's line of sight.

The second application bean, called Flatscape, is developed using Java2D. An example snapshot is shown in Fig. 11. The operations on the Glyph tree in Flatscape are:
Op_1: creates simple geometric figures (line, rectangle, ellipse, polygonal line, spline) and bitmap images. The figures can be grouped into PolyGlyphs, as in the case of the room object.
Op_2: deletes any of the above listed Glyphs
Op_3: sets Glyph properties, such as contour color/thickness/dash, fill color, position, dimensions, visibility, and manipulation constraints (e.g., movement only along the x-axis). The lines also have arrow style and the splines have other properties. The user can also apply 2D affine transformations to Glyphs.

The telepointers in Flatscape are 2D arrows pointing away from the center of the room towards the part of the room currently viewed by the user. This is different from the typical telepointer, e.g., that found in [5], which only shows position and not orientation. In our 2D implementation the user who owns the telepointer, rather than the recipient, has control over its visibility.

B. Human Factors Experiments

Although the WYSIWIS idealization recognizes that efficient reference to common objects depends on a common view of the work at hand, strict WYSIWIS was found to be too limiting and relaxed versions were proposed to accommodate personalized screen layouts [20]. In subsequent work [21], problems were reported with non-WYSIWIS systems because manipulation and editing processes were private and only results were shared. This discontinuity of the interaction created ephemeral environment differences that affected collaboration. Our essential principle for heterogeneous collaboration is that every user's action is interactively and continuously reflected in other users' workspaces, with a varying degree of accuracy or realism or through a qualitatively different visualization. Thus, the shared reference is maintained.

To study the effect of heterogeneity on human performance we used the above two applications (2D Flatscape and 3D cWorld). The participants are instructed to use these applications as a tool to decide where they would like to have the moving company place their furniture when it is moved to their new office. The study was a within-teams design, that is, collaborative teams were assigned to each of four conditions, 2D→2D communication, two 2D↔3D communication arrangements and 3D→3D communication. We force the users to collaborate by having each user ask the other participants to perform the user's furniture placement. Each user may also verbally request repairs to the placement, such as "Move the desk a bit closer to the window." We expected the heterogeneous collaborations to experience more repair statements, and to take somewhat more time than the homogenous collaborations. We expected fewer repairs in the homogenous tasks because synchronized views give the collaborators equivalent information.

We found that there was little relationship between the number of repair statements and the time it took to perform the task. Because of this, we felt comfortable in separating out the number of communication repairs and the object placement times in our multivariate data collection and running a Student *t*-test that compared heterogeneous to homogenous collaboration environments. Our results are shown in Table 1.

Table 1. A comparison of communication repair and task performance time for homogeneous vs. heterogeneous collaboration environments.

	Heterogeneous 2D-3D & 3D-2D	Homogenous 3D & 2D	Student *t* test results
Mean time / std. dev.	99 / 116	94 / 116	$t = 0.337$, $p = 0.37$
Mean # repairs / std. dev.	2.37 / 2.35	1.97 / 2.34	$t = 1.337$, $p = 0.18$
No. of objects	188	92	

The two Student *t*-tests show no significant differences between the heterogeneous and homogenous conditions.

We examine this explanation further by looking at the 2D→3D and 3D→2D tasks. In the 2D→3D task, the person giving directions was in the 2D environment and the person performing the object placement was in the 3D environment. This was reversed in the 3D→2D task. If the difficulty were with the 3D environment, we should find that the average object placement time is longer in the 2D→3D case. The average time difference was significantly longer ($t = 5.149$, $p < 0.001$). Table 2 displays the results of our post hoc comparisons.

Table 2. A comparison of the communication repair and task performance time between 3D→2D and 2D→3D object placement requests.

	3D→2D	2D→3D	Student *t* test results
Mean time / std. dev.	79 / 128	121 / 99	$t = 5.149$, $p < 0.001$
Mean # repairs / std. dev.	1.26 / 1.15	3.53 / 2.71	$t = 1.746$, $p = 0.086$
No. of objects	47	45	

We also note that the mean number of repairs is considerably less for the 3D→2D than the 2D→3D object placement. This result is tending towards significance ($t = 1.746$, $p = 0.086$). We interpret this effect to arise for two reasons. The 3D view allows the team member to give better directions to the 2D member and the 2D application is easier to perform manipulations in. This suggests that heterogeneous collaboration is better than homogeneous collaboration as long as it is 3D→2D since the 3D user is better "immersed" into the virtual world. These preliminary data are supportive of the typical scenario envisioned for heterogeneous collaboration, where an expert from a high-end desktop environment assists a mobile fieldworker with a low-end portable access device.

VII. Conclusions

Information exchange is a hallmark of productive human activity. Global networking promises expanded capabilities. But a major obstacle to mass deployment is the widely disparate computing environments often used. The research presented here addresses this mass utility issue and develops a system design to accommodate heterogeneities.

We present a data-centric framework for synchronous collaboration of users with heterogeneous computing platforms. Our approach allows clients with different computing capabilities to share different subsets of data, e.g., due to compression in order to conserve bandwidth. XML, which serves as the communication medium, is a standard that has already gained wide acceptance, and

provides a powerful medium for both data exchange and visualization specification. Other unique features of the DISCIPLE framework include simultaneous support for collaboration-aware and collaboration-transparent applications and an interface that is open and customizable with respect to the context in which it is placed and for particular user needs.

Our experiments with wireless awareness show that applications can adapt to the wireless networking environment proactively and reactively by deploying the awareness scheme. The experimental results demonstrate that wireless-aware applications can achieve a near-optimal data-volume transmission. As a result, the application can use the network and computing resources in an efficient manner, and thus yield improved performance. This research provides a feasible scheme to enhance the quality-of-service for wireless multimedia applications in heterogeneous networks. Although the experiments are conducted in a wireless LAN, the wireless awareness architecture and associated methods are extensible to more complex communication environments.

The solution presented can be extended to a broader domain of data sharing in heterogeneous environments. Dearle [22] discusses the problem of how to maintain a persistent user's desktop state for mobile users. The problem is similar to asynchronous collaboration of multiple users, except that here the user "collaborates" with himself/herself at different times. The problem can be generalized, so that the user can resume work at a different platform, with a different display device. For example, a user may carry his/her XML documents (models) on a wearable computer. As the user connects the computer to different display devices, heterogeneous views may be created according to the device characteristics.

Further information about the DISCIPLE project and heterogeneous collaboration is available at:

http://www.caip.rutgers.edu/disciple/

ACKNOWLEDGMENT

Research contributors to this project include Professors James Flanagan and Marilyn Tremaine. Graduate researchers involved in the project, Allan Krebs, Bogdan Dorohonceanu, Liang Cheng, Mihail Ionescu, and Helmuth Trefftz were critical to the implementation and evaluation of the concepts presented here. The research is supported by NSF KDI Contract No. IIS-98-72995 and DARPA Contract No. N66001-96-C-8510 and by the Rutgers Center for Advanced Information Processing (CAIP) and its corporate affiliates.

REFERENCES

[1] Digital Sandbox, Inc., The miniGL Project, http://www.dsbox.com/minigl.html

[2] I. Marsic, "DISCIPLE: A Framework for Multimodal Collaboration in Heterogeneous Environments," *ACM Comp. Surveys*, vol.31, no.2es, Jun. 1999.

[3] D. Garfinkel, B. C. Welti, and T. W. Yip, "HP SharedX: A Tool for Real-Time Collaboration," *Hewlett-Packard J.*, vol.45, no.2, pp.23-36, Apr. 1994. http://www.hp.com/hpj/94apr/toc-04-9.htm

[4] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks, "Rendezvous: An Architecture for Synchronous Multi-user Applications," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW'90)*, Los Angeles, CA, pp.317-328, Oct. 1990.

[5] M. Roseman and S. Greenberg, "Building Real-Time Groupware with GroupKit, a Groupware Toolkit," *ACM Trans. Computer-Human Interaction*, vol.3, no.1, pp. 66-106, Mar. 1996.

[6] P. Dewan and A. Sharma, "An Experiment in Interoperating Heterogeneous Collaborative Systems," *Proc. 6th European Conf. on Computer Supported Cooperative Work (ECSCW'99)*, Kluwer Acad. Publ., Copenhagen, Denmark, pp.371-390, Sep. 1999.

[7] D. R. Olsen, S. E. Hudson, M. Phelps, J. Heiner, and T. Verratti, "Ubiquitous Collaboration via Surface Representations," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW'98)*, Seattle, WA, pp.129-138, Nov. 1998.

[8] Wireless Application Protocol Forum, "Wireless Markup Language Specification," SPEC-WML-19990616.pdf. At: http://www.wapforum.org/what/technical.htm

[9] WearLogic web page: http://www.wearlogic.com/

[10] P. Dewan, "Architectures for Collaborative Applications," in *Computer Supported Co-operative Work*, M. Beaudouin-Lafon (Ed.), John Wiley & Sons, Chichester, England, pp.169-193, 1999.

[11] W. Wang, B. Dorohonceanu, and I. Marsic, "Design of the DISCIPLE Synchronous Collaboration Framework," *Proc. 3rd IASTED Int'l Conf. Internet, Multimedia Systems and Applications*, Nassau, Bahamas, pp.316-324, Oct. 1999.

[12] B. Dorohonceanu, B. Sletterink, and I. Marsic, "A Novel User Interface for Group Collaboration," *Proc. 33rd Hawaiian Int'l Conf. System Sciences (HICSS-33)*, Maui, Hawaii, Jan. 2000.

[13] M. E. Fayad and S. C. Schmidt, "Object-Oriented Application Frameworks," *Comm. ACM*, vol.40, no.10, pp.32-38, Oct. 1997.

[14] Sun Microsystems, Inc. JavaBeans API Specification, http://www.javasoft.com/beans/

[15] L. Cheng and I. Marsic, "Wireless Awareness in Heterogeneous Networks," Submitted for publication.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman, Inc., Reading, MA, 1995.

[17] R.L. Carter and M.E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," Technical Report BU-CS-96-006, Computer Science Department, Boston University, Mar. 1996.

[18] IBM, Inc., Bean Markup Language (BML), http://www.alphaWorks.ibm.com/tech/bml/

[19] D. Comaniciu, P. Meer, D. Foran, and A. Medl, "Bimodal System for Interactive Indexing and Retrieval of Pathology Images," *Proc. 4th IEEE Workshop on Applications of Computer Vision (WACV'98)*, Princeton, New Jersey, pp.76-81, Oct. 1998.

[20] M. Stefik, D. G. Bobrow, S. Lanning, and D. Tatar, "WYSIWIS Revised: Early Experiences with Multiuser Interfaces," *ACM Trans. Information Systems*, vol.5, no.2, pp.147-167, Apr. 1987.

[21] D. Tatar, G. Foster, and D. G. Bobrow, "Design for Conversation: Lessons from Cognoter," *Int'l J. of Man-Machine Studies*, vol.34, no.2, pp.185-209, 1991.

[22] A. Dearle, "Toward Ubiquitous Environments for Mobile Users," *IEEE Internet Computing*, vol.2, no.1, pp.22-32, Jan./Feb. 1998.