Ground Concepts and Tools of *CSCW* *

[Current Methodologies and Future Trends] † .

Ahmed S. Mostafa ‡ §
University of Fribourg
Department of Informatics
Switzerland
ahmed.mostafa@unifr.ch

ABSTRACT
The paper provides a brief discussion of current methodologies incorporated in Computer-Supported Cooperative Work (CSCW) by discussing 3 different implementations of Groupware systems. The paper also gives an insight about supporting Groupware in windowing systems as well as discussing some of the challenges facing future implementations of User Interface Build Tools.

General Terms
Computer Supported Cooperative Work, Human Computer Interaction, Groupware, Ubiquitous computing, Windowing Systems

Keywords
CSCW, HCI, GWWS, SDG, Roomware

1. INTRODUCTION
Collaboration between individuals and groups is extremely valuable to achieve common goals and complete tasks. However, there are many challenges that face collaborators; whether they are communicating physically or remotely.

For remotely located teams, the challenges are bigger, starting from technical challenges such as connectivity, interoperability, persistence and security to financial challenges such as providing an effective solution with relatively acceptable cost. There are more challenges such as the effort spent in educating teams on new collaboration techniques. All of these challenges may discourage teams and organizations from adopting such new technology or solution.

For physically located teams, there are different techniques and methodologies that enable collaboration, some of these methodologies are manual or using simple computer technology; such as black or white boards, or manual Scrum and Agile techniques. There are other methodologies which involve more high-technology such as smart boards, shared electronic surface screens, and Scrum and Agile software which can help collaboration and increase productivity.

Despite of the above team types, there are common challenges that face both types such as technical challenges and cost of implementation. For example, the cost of installing, implementing, and supporting a new collaboration device may not encourage organizations and companies of adopting them. Also, even with an affordable cost and support plans, such tools might be abandoned because of their hard learning curve, or may be just because it lacks a single feature that means a lot to collaborators which cause them reject it altogether. Therefore, the technology should increase productivity and to be easy to learn and apply in a way that motivates the team to use it. Another important aspect is the extensibility of the technology to add more features or extensions, which helps it to become one of the main collaboration methods inside the organization.

The following section provides a brief discussion about 3 different Groupware frameworks and toolkits namely the *Rendezvous* architecture framework, the *BEACH* application model, and *GROUPKIT* for building real-time conferencing applications. Next, the paper briefly discusses the *Groupware support in the Windowing System* and its benefits. The last section in the paper will list some challenges facing future implementations of user interface software tools.

2. *GROUND CONCEPTS AND TOOLS*
There are several research papers and implementations for Groupware toolkits and frameworks. Each Groupware toolkit provides certain advantage/disadvantage over the other. From concepts perspective, Groupware can be classified in 3 categories: [2]

*Seminar work is under supervision of Dr.Denis Lalanne, Dr. Maurizio Rogamonti, and Dr. Bruno Dumas.

†Paper can be downloaded from CSCW Fribourg's university homepage: `http://diuf.unifr.ch/diva/web/site/index.php/teaching-seminars/10-seminars/141-enhancing-collaboration`

‡Ahmed S. Mostafa is a computer science student in the Swiss Joint Master Program of Fribourg, Bern, and Neuchatel Universities. For more information, please go to: `http://mcs.unibnf.ch.`

§Ahmed S. Mostafa is a Senior Consultant in TMNS. For more information about professional experience, please go to: `http://www.linkedin.com/in/ahmedsmostafa.`

- **SDG toolkits and applications:** Single Display Groupware denotes applications that run on a single host computer but allow multiple users to interact with the application simultaneously. They require support for multiple input devices and often provide additional features.

- **Distributed Groupware:** Several networked hosts computers share an application, they require synchronization and sharing of applications over the network and has to handle race conditions caused by network delays. Each host computer has one pair of input devices; a keyboard and a mouse. They allow one user to interact with a shared application at a time.

- **Remote Input Devices:** Send high-level events to control a mouse and keyboard on a remote computer. They are usually used to control different host computers with a single keyboard and a mouse. However, they support interaction with a single computer at a time.

2.1 The *Rendezvous* Architecture for Multi-User Applications

When people have meetings or discussions, they frequently use *conversational props*; such as physical models, drawings, or other concrete representations of information used to enhance the exchange of information [1], the challenge to make use of such *conversational props* becomes harder when participants are geographically separated. Voice and video conferencing have made important progress in enabling collaboration between remote teams but still has some limitations that need to be solved such as single manipulation of conversation, or forcing all participants to see or review the same information just because one of the participants requested it [1].

The *Rendezvous* architecture and language are designed to provide applications that can be used as conversational props which helps remote teams to collaborate. There are several benefits of using *Rendezvous* on top of voice and/or video conferencing [1], such as:

- The flexibility it provides to customize views for each participant; in a sense that each participant can see or review physical props or notes without forcing other collaborators to view it.

- Enabling teams to collaborate and share ideas on the same electronic space.

- Floor control to force collaborators to take turns (e.g. turns in a game).

- Support for view replication for all or part of the shared view space.

- Support for user inputs and actions; such as joining, resetting, or quitting a game or a conference. Also, direct and indirect manipulation of items is supported; such as dragging cards or boards.

Rendezvous can be considered as a User Interface Management System (UIMS) because it provides functionalities of creating custom user interface components while being isolated from physical OS-related graphics layers [1]. The framework is efficiently designed by a notion of consistency and dialog independence with shared abstraction stores which allow users to have independent views while still being able to create more different views with consistent information. This is done by utilizing different object oriented patterns together to compose a new pattern which is known as Abstraction-Link-View pattern [1].

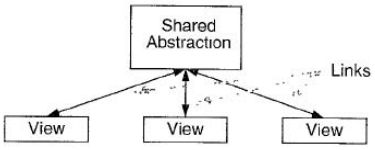

Figure 1: Abstraction-Link-View structure

The object model is extended with a declarative graphics model for generating and maintaining displays, an event system for reacting to user inputs, and a constraint maintenance system to help maintain general states. *Rendezvous* is built on the Common Lisp Object System, and its class hierarchy is a directed acyclic graph with more than 350 reusable classes [1].

2.2 *BEACH* Application Model for Synchronous Collaboration

BEACH is a software infrastructure framework concerned about collaboration in ubiquitous environments. It provides the functionality for synchronous cooperation and interaction with heterogeneous environments and *roomware components* which are room elements with information technology integrated and enabled such as interactive tablets, walls, or chairs which is very beneficial for building CSCW applications.

BEACH stands for: **B**asic **E**nvironment for **A**ctive **C**ollaboration with **H**ypermedia [5]. Beside the software development techniques used to build the framework ensure its reusability and extensibility, Its conceptual model for synchronous ubiquitous computing is a result of combining several related research areas which are closely related to each other: [5]

- **Human Computer Interaction (HCI)**: is a broader research area which deals with user interfaces and interaction techniques.

- **Ubiquitous Computing**: explores dynamic environments with heterogeneous devices such as smart walls, pads, and fabrics, etc. . .

- **Computer-Supported Cooperative Work (CSCW)**: offers techniques to handle synchronous collaborative human interaction with technology enabled devices and services.

The above research areas contributions can be formed into *BEACH*'s design as three design dimensions: [5]

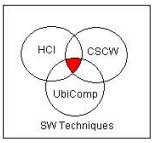

Figure 2: Collaborative ubiquitous computing applications

1. *separation of concerns* to five separate models: Data, Application, UI, Environment, Interaction models. This separation allows for context awareness which is essential in CSCW.

2. *coupling and sharing* of models and information between collaborating devices and users; complexity of a system can be reduced by sharing certain models among collaborating devices/users; e.g. sharing the data model for documents reduces the complexity in dealing with distributed applications...

3. *level of abstraction* which is an already known software engineering technique to reduce complexity of designed layers; this technique can be also applied in CSCW systems to reduce complexity and allow reusability.

The implementation of BEACH is fully object-oriented by VisualWorks Smalltalk. It supports different user-events as well as integration with multiple devices. It is also optimized to reduce the amount of data transmitted via network in order to minimize bandwidth usage [5].

2.3 *GroupKIT* Toolkit for real-time conferencing applications

GroupKIT is a real-time conferencing framework for building groupware toolkits and components. *GroupKIT* is implemented based on a specific set of design requirements. For example, support for gesturing, graphical annotation, floor control policies, and assisting late comers to get up to speed. This is beside technical-related requirements that are mainly programmer's responsibility; such as managing the conference and any sub activity initiated during the conference (joining or leaving of participants...), and the possibility to persist/save sessions for later reviews... [4].

GroupKIT's infrastructure is based on replicated architecture and communications support which supports conference initiation, registration and subsequent communications owned by conference participants[4].

GroupKIT is based on three components: the runtime architecture for running and managing the distributed processes, transparent overlays for adding general groupware components (such as drawing multiple cursors for different users on an application's graphic), and open protocols for extending the basic communication to fit the needs of new applications [4] [1].

GroupKIT's goal is similar to the *Rendezvous* system [1]. However the design and architecture paradigm is different as listed in the previous paragraphs. Additionally, *GroupKIT* does not support a shared public graphic model as the *Rendezvous* system does [1]. *GroupKIT* is implemented in C++ and based on another system called *The Interviews toolkit* [1] [4].

3. GROUPWARE SUPPORT IN THE WINDOWING SYSTEM

Many Groupware toolkits do not allow the execution of multiple Single Display Groupware (SDG) applications simultaneously, running different applications that rely on different toolkits at the same time is complicated or even impossible[2]. A Groupware Windowing System (GWWS) for SDG has some advantages such as executing legacy applications in a multi-user context and simultaneously with other SDG applications. Technically, the X Window System can be enhanced to support multiple independent mouse cursors, which can operate in multiple applications. Such enhancement includes support for floor control mechanism and each GUI could have different access rules [2].

GroupKIT and *Rendezvous* rely on applications being built against the toolkit's specific API. None of these tools supports a fully ad-hoc per-device configuration to assign a specific device to a specific cursor at any point in time [2]. Groupware Windowing Systems should provide support for the following domain-independent features: [2]

- Arbitrary number of input devices.
- Simultaneous interaction of multiple input devices.
- Fine-grained floor control.
- General purpose graphics annotation overlays.

Integrating Groupware support into a Windowing system keeps application's compatibility intact while enabling multiple input devices to use it. As applications deal with a mouse and keyboard as "stream events", the semantic of input-devices events will be preserved and only the Windowing system will handle the low-level physical interactions with the input devices. Such modifications to the windowing system can be ideally placed into the **Window Manager** instead of placing it into any Groupware toolkit [2]. Although programming on the Window Manager level has its advantages as described earlier, it tends to be time consuming and tedious. Further, when each programmer creates all interface components from scratch, it is practically impossible to provide much widespread consistency for the user [3].

Windowing support for Groupware is extremely beneficial for CSCW as it enables collaboration on different applications and tools supported by isolated windows, which helps in utilizing the most features of every running application simultaneously[1].

[1]An implementation and a proof of concept is implemented within the HxI Initiative Project [braccetto][2] which contains three main components: a Multi-Pointer X Server (MPX), a Multi-Pointer Window Manager (MPWM), and a DeviceShuffler which supports many-to-many mouse to cursor configurations for a GWWS system [2]

4. IMPLEMENTING FUTURE USER INTERFACE SOFTWARE TOOLS

User Interface Software Tools are essential for many reasons: they help achieve a consistent look and feel since all user interfaces created with a certain tool will be similar, they also allow user interface to be created more quickly which helps in rapid prototyping, and reduce the amount of code that programmers need to produce when creating a user interface [3].

Determining the successful approaches for implementing future user interfaces is very important for CSCW implementations and it requires thorough analysis for the past user interface tools. For example, User Interface Management Systems (UIMS) were not well-accepted because of it is high-level of abstraction between users and details of input and output devices, it is extremely valuable to control the low level pragmatics of how the interactions look and feel. Also, the standardization of the user interface elements on the desktop paradigm made the need for UIMS unnecessary [3]. A new phase of increased experimentation and change in user interfaces is coming in the future because of different factors, which make the moving-target problem[2] a serious issue again: [3]

- *Computers becoming a commodity*: Computers available to the public are often as fast or faster than the researcher's, this may have a profound impact on how and what computer science research is performed; which will result in qualitative change in the kinds of user interfaces possible which requires to contain visual and audio effects.
- *Ubiquitous computing with various input and output capabilities*: The concept of user interfaces provided to users will need to be adapted and improved in which it could interact with many different types of computerized systems: mobiles phones, digital pager, desktops and surface tables... which have different or no input devices.
- *Recognition-based user interfaces*: With new research results appear in various computer-science fields; user interfaces will need to include new ways of inputs and output devices such as utilizing gestures, touch, and speech recognition and related technologies in communicating and collaborating with systems or teams.
- *Three-Dimensional based user interface*: Applications that may benefit from three-dimensional interfaces include simulation and training as well as interactive exploration of complex data environments or as entertainment tour guides in museums. Providing three-dimensional user interfaces offer significant opportunities for human-computer interaction. However, it is still a difficult problem due to standardization, optimization...

Also, there still a continuous effort needed for providing suitable user interfaces that fit needs of disabled, younger non-experienced, or even left-handed persons. For example "most of user interface toolkits and design knowledge has implicit assumptions that users are "sitting" and have "two hands" available while operating" [3]. Also, the positioning of arrows in current keyboards or the shape of most of new mouse devices do not fit left-handed persons. CSCW applications needs user interface software tools that provide a rich context information about the user, devices, and the application's state, rather than around event which will enable providing "replaceable user interfaces for the same applications to provide different user experience on different devices" [3].

[2] *Moving Target Problem:* It is difficult for UI build tools to keep pace with rapid development of new UI techniques and technology [3].

5. FUTURE WORK

Some of the above mentioned issues need to be handled in the operating system and windows manager level, and the remaining issues could be handled in the user interface toolkit level. However, it is an interesting challenge to provide clear standards for such low level requirements. The work here is introductory and still need to do more research about how distributed user interfaces can be implemented in a more flexible way in order to cover the limitations and concerns listed above. The focus could be in implementing a low-cost groupware toolkit or solution while in the same time keeping the learning curve acceptable to developers and software organizations to adopt.

6. CONCLUSION

This paper provided a brief outline of currently implemented groupware toolkits and solutions, with an insight of how it can be supported in windowing systems. The paper also outlined the challenges that might face implementing future user interface software tools; which is very important for researchers and developers to be aware of due to the fast growing ways of communication between computers and humans. As seen earlier an ultimate toolkit/solution can be obtained from a careful revision of past researches in order to grasp the best of each system in a way that overcomes certain challenges.

7. REFERENCES

[1] R. D. Hill, T. Brinck, S. L. Rohall, J. F. Patterson, and W. Wilner. The *Rendezvous* architecture and language for constructing multiuser applications. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125, June 1994.

[2] P. Hutterer and B. H. Thomas. Groupware support in the windowing system. *Australian User Interface Conference (AUIC2007), Research and Practice in Information Technology (CRPIT)*, 64:39–46, 2007.

[3] B. Myers, S. E. Hudson, and R. Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7(1):3–28, MArch 2000.

[4] M. Roseman and S. Greenberg. Groupkit: A groupware toolkit for building real-time conferencing applications. *ACM CSCW Conference Proceedings*, pages 43–50, November 1992.

[5] P. Tandler. The *BEACH* application model and software framework for synchronous collaboration in ubiquitous computing environments. *Submitted to JSS, special issue on UbiTools*, 4.2(1):1–35, February 203.