Architecture for Synchronous Groupware Application Development

Roland Balter, Slim Ben Atallah, Rushed Kanawati


This paper describes the design choices and the prototype implementation of CoopScan, a generic framework for synchronous groupware development. We focus on architectural issues and on strategies for integration of existing single−user applications into a collaborative environment. In this work, we propose a generic approach to application re−use. This approach is validated through the development of a testbed synchronous collaborative editor.

1 INTRODUCTION

Multiple criteria are proposed to classify groupware applications [7]. The most commonly used ones are the interaction scheme (synchronous versus asynchronous), the architecture (centralized, replicated or hybrid) and the strategy for building shared applications by integrating existing single−user ones.

Focusing on the development of synchronous groupware environments, this work presents a study of architectural aspects and development strategies to building shared applications. We propose a generic approach to build synchronous multimedia CSCW (Computer Support Cooperative Work) applications for teleconferencing environment. We are also interested in developing flexible access control protocols to shared information as well as providing robust protocols able to handle dynamic site connection and fault tolerance.

This paper contains two major descriptive sections and a conclusion. In Section 2 we review the groupware architectures and various strategies for building shared applications. In Section 3 we present the CoopScan architecture. The last section presents some conclusions and future research directions.
2 GROUPWARE ARCHITECTURES AND DEVELOPMENT STRATEGIES

2.1 Groupware architectures

Two architectures are usually considered in computer science literature: the centralized scheme and the replicated scheme. In the centralized scheme, only one instance of the shared application runs on a given site. All user inputs to the shared application are forwarded to this single instance while the outputs are broadcasted back to different user displays. An implementation example of this scheme is Rendezvous system [8] that is based on a client−server architecture with all user interactions and display management handled by the server. Each user has an associated view process that interprets input events and display directives. The main advantage of such a client server architecture is the low implementation cost. Besides, consistency between user’s views (Section 2.3), and dynamic connection of sites during a work session are also easier to support. However, this architecture scheme has some drawbacks such as high vulnerability to site failures, congestion problems and increased response time.

In a replicated architecture, each user executes an instance of the shared application. Inputs are distributed from the user window to all instances. The output from each copy is delivered only to the local window system. This scheme answers all the drawbacks of the centralized one. Its main advantages are high robustness, and short response time. In addition, it better supports heterogeneity and view customization [9]. However, the main drawback of this scheme is hard consistency management. Several toolkits have adopted this strategy. Examples are MMconf [6], and GroupKit [13].

A third architecture scheme, called the hybrid architecture, is derived from the last two ones. This scheme is proposed to overcome the disadvantages of both mentioned models. The idea is to replicate user interfaces connected to a central application process. User interface operations are performed locally and all other tasks are performed by the centralised application process.

2.2 Integration Schemes for Shared Application Development

Two approaches for building shared applications are commonly discussed. The first one consists of developing the whole application from scratch. Such an approach is usually adopted in case of specific applications that are extremely
hardware dependent. An example is an industrial process controller with strict quality of service (QoS) constraints. The second approach is based on a re-use strategy in which already existing single user applications are integrated in a cooperative environment. In the latter approach two models of integration are distinguished:

**Low level integration scheme**

In synchronous groupware, users sharing a common information space are permanently aware of all actions on this space. The awareness is implemented through an event exchanging mechanism. In the low level scheme, already existing single-user applications are integrated into a cooperative environment using a mechanism based on exchanging window system events [1]. A suitable handling mechanism intercepts data flows between the application and the window system. The intercepted events are then broadcasted to all remote users sharing the information space. The major advantage of this approach is that it permits transparent integration of already existing applications, without any modification. However, it presents some drawbacks such as the high cost development of control protocols. In fact, most protocols that ensure consistency between user views are based on event journalization mechanisms. Adopting the low level approach, the exchanged events between user’s views always summarize elementary user actions on shared information, for example, `XButtonEvent` in X-Windows environment. These events have generally a poor semantic meaning; they do not contain enough structured information on the user’s actions. This makes it difficult to develop flexible control protocols for solving data access contention, and dynamic join and leave operations [1]. Another drawback is the problem of display customization (view customization) and high network traffic.

**High level integration scheme**

In this integration scheme, the cooperation mechanisms are based on application events. An application provides a well defined interface to its environment. The user actions on the shared information are encapsulated into application events and then delivered to the interface layer. A callback mechanism detects when actions on the shared information occur.

The high level integration scheme allows low traffic network and flexible data access control. However, this model of integration can only be applied to a restricted class of open applications. An open application provides an API (Application Programming Interface) mechanism through which it can be driven by external modules, and not only by its user interface. Such an application offers also a callback mechanism that provides an awareness tool for notifying user actions on the application objects.
2.3 Interaction modes in groupware applications

A widely adopted taxonomy of groupware applications is that based on the temporal nature of interactions among participants [4]. The interactions can be achieved in a synchronous or an asynchronous way [9]. In synchronous groupware, a number of users simultaneously manipulate the shared information, which can be for instance, a document, a draw panel or a daily schedule. Each of these users has its own view of the information [2]. In the synchronous way, modification on the shared information must be reflected immediately in all views for all users. In this way, consistency between views is ensured at any time. The user views are not necessarily identical. For example, two users can manipulate different sections of the same document, in this case, coupling among the shared space is said to be loose. When the views are identical, coupling among views is said to be tight (WYSIWIS – What You See Is What I See) and temporal constraints become strict [10].

3 COOPSCAN DESIGN

The architecture scheme is not specific to groupware applications but common to all distributed and communicating systems. However, in synchronous groupware, temporal constraints such as response time and delay jitter are fundamental criteria for choosing a convenient architecture scheme. In our current work, we focus on building synchronous groupware with WYSIWIS interaction mode. For such groupware applications, response time is a fundamental constraint especially in a wide area network configuration. We have then chosen a fully replicated architecture scheme for CoopScan.

CoopScan aims at providing a generic framework for developing synchronous CSCW applications. Generic aspects imply providing generic control services that allow flexible data access protocols and robust leave and join mechanisms. In order to provide such mechanisms, the high level integration scheme is adopted. This choice is mainly motivated by the low cost development of control protocols. In fact, in this approach, the events delivered by the application are of a high semantic level compared to those delivered by the window system. These events summarize the complex user actions on the shared information, for example: Open (document), Insert (element, document), allowing easy identification of both, user actions and manipulated data.

However such an integration scheme can only be applied to open applications. To overcome this limitation we argue that the paradigm of structured active multimedia document (SAMD) can be employed as basic concept for building platform support for synchronous groupware. The SAMD provides two essential properties for
building synchronous groupware with flexible control protocols; a structured information space and an active data management. The structured information eases the development of flexible data access and concurrency management policies.

The active data management consists of notifying the environment of the application of all actions performed on the shared information. This property is a main vehicle for implementing the awareness mechanism. The active data also provides an API mechanism. Actions on application data are not only performed through the user interface, but also through the API. External modules can then control the application.

SAMD paradigm for building generic platform suits well with the needs of synchronous applications mentioned above. Adopting this approach, a single−user application is linked to an active document. All application objects are then linked to specific document elements. To control an application, a user edits a document whose elements represent objects handled by the application. When operations are performed on the document elements, application performs corresponding actions on its specific objects. To summarize, the application user−interface will be represented by a document that has well defined abstract structure suitable for the application needs.

**CoopScan toolkit**

CoopScan architecture is described through an abstract object−based model. Three main layers are distinguished: the application layer, the communication layer and the conferencing layer

The application layer is the existing single−user application.

The communication layer encapsulates the transport protocols.

The conferencing layer provides the necessary mechanisms to initiate, to join and to control a collaboration session. This layer can be described with an abstract active object model called agent model:

- the Local Agent (LA_g) represents a given user during a work session. The LA_g executes control protocols such as shared information access, floor passing negotiation and user’s roles assignment. User’s actions on the shared information are authorized by LA_g. The events resulting from user’s actions are hence broadcasted to all nodes. LA_g encapsulates application events into a well defined communication structure delivered to the the communication layer.

- the Distant Agent (DA_g) processes all LA_g messages on the site it is located on. First, DA_g receives a communication structure from the communication
layer. It decodes the message and then determines the right API function to invoke.

- **the Session Agent (SAg)** manages all session information such as users (rights, location), and applications (documents, access lists, status). The session agent is responsible for the initialization of the work session and all dynamic aspects of joining and leaving the groupware.

In order to validate the proposed framework, a synchronous cooperative editor using an already single−user SAMD [12] is developed. We have also integrated in CoopScan architecture a free ware audio tool. The current prototype is developed on a SunOs platform in C language. The communication layer consists of a point to point and a broadcast communication services using TCP/IP protocols with BSD Unix Sockets.

Users share a **common information space** that consists of a set of user documents, in addition they can define their own personal work space. Edition operations between the two spaces are allowed. User’s views of the shared information are coupled in a WYSIWIS mode.

Two approaches for **initiating a session** are possible: a user centered approach and a data−centered approach. In the user centered approach, a shared context is first initiated by set of users. Documents are subsequently introduced in this shared context. In the data−centered approach, a CSCW session is automatically initiated when a user opens a shared document already manipulated by another user. Currently, the user−centered approach is implemented, however, we plan to implement the data−centered one in the near future.

Users can dynamically join and leave a work session. Participation is sponsored by a designated site. When a late comer arrive, the sponsor site provides him with the current session context (participants and their status, the set of open documents). This is achieved by simple file transfer based operation.

A public domain audio conferencing tool, the VAT tool [3], is integrated in the prototype in order to allow the users to exchange audio comments in a real time manner.

Three roles are defined: the chairman (session manager), the editor (W/R operations) and the presenter (read only). Concurrency management is done by applying a floor passing policy. Different policies are provided (Designation, FIFO, round robin). It is up to the chairman to decide which policy to apply.
4 CONCLUSION

In this work, we are interested in providing a generic framework for the development of synchronous groupware applications. We study the requirements to develop such a framework and to provide flexible control protocols. We focus in this study on both architectural aspect and integration schemes of existing single user applications in a cooperative environment. A high level integration scheme is adopted. As this approach is reserved to restricted class of open applications, we introduce the SAMD as a basic concept for developing synchronous groupware. Following this approach, we have developed a synchronous collaborative edition application. However, the described framework may also be used to integrate X–window based application.

Future works

Two main research activities are initiated. The first one deals with access control protocols and role designation mechanisms. The second one focuses on dynamic join and leave protocols with temporal and consistency constraints. We will also focus on QoS requirements such synchronization constraints between several multimedia data flows, and event ordering in synchronous groupware.

ACKNOWLEDGEMENTS

This work is partially supported by the CNET France (Centre National d’Études de Télécommunications) under the convention 911B. We would also thank the members of OPERA–INRIA team for their precious collaboration.

REFERENCES


Bibliography


