Adaptive Consistency Management Support for Limited Capacity Devices in Ad-hoc Mobile Networks

Jim McGovern and Caspar Ryan
RMIT University
School of Computer Science and Information Technology
RMIT University, Melbourne, Australia
jim.mcgovern@rmit.edu.au, caspar.ryan@rmit.edu.au

***Abstract*—With the advent of 3G infrastructure and the Next Generation Internet [1], wireless communication and mobile computing devices provide the opportunity to undertake complex information processing tasks, such as real-time distributed collaboration, "on the move". Consequently, a number of collaborative systems have been developed to deal with the limited capacity and heterogeneity of small devices, and the lower bandwidth and latency inherent in wireless communication. However, for the most part, these systems have relied on a central server and/or explicit application re-configuration in order to accommodate the limited resources of mobile devices.**

Aiming to provide a more flexible alternative, this paper addresses the implementation of complex real-time synchronous editing in ad-hoc mobile networks built purely on peer-to-peer communication among small devices. The chief premise is that software components running locally in the ad-hoc network must adapt their collaboration algorithms to work within the processing and storage constraints of the participating mobile devices. As a result, this paper describes a mechanism for dynamically modifying the level of collaboration support for a group working on a shared real-time editing task, within an ad-hoc mobile network. This mechanism has been partially implemented and tested in a prototype system called MOCORA (MObile COllaboration in Replicated Architecture).

***Index Terms*—mobile collaboration, real-time collaboration, groupware, heterogeneous devices, replicated architecture, application adaptation, 3G applications, Next Generation Internet.**

I. RATIONALE - THE SIGNIFICANCE OF REAL-TIME MOBILE COLLABORATION

Real-time collaborative applications support two or more users working on a shared information processing task over computer networks. This work can vary from simple communication of messages, to real-time collaboration on a shared workspace with the aim of producing a single, agreed artefact. The shared workspace can include combinations of text, graphics, or real time media such as video and audio, with specialized applications involving the editing of complex, highly constrained and semantically rich diagrams such as those used in software engineering [2].

Collaboration using mobile devices has the potential to support work-based applications such as medical telediagnosis, distributed mobile offices, construction work and field workers requiring links to a central office; as well as military, search and rescue and national emergency operations [3-5]. Mobile devices can also potentially support recreational collaboration activities such as bush-walking (hiking) groups [5], hunting groups [6] and motor cyclist (biker) communities [7].

Some of these applications may be able to access central servers, but others may necessarily require direct access to their peers only. Consider, for example, the case of a major disaster in which search and rescue operations may require teams to enter areas completely formed as a result of the disaster. In such a situation, rescuers must avoid dangerous obstacles and find their way through unfamiliar surroundings, to rescue injured or trapped people. Such a scenario is likely to render infrastructure ineffective, and communication may be restricted to peer-to-peer communication using small wireless devices. Similar scenarios can also be readily described for military applications.

In developing such applications for mobile environments, a wide range of devices with vastly different processing power and storage capacity may need to be supported. Mobile devices can range from laptop computers with the equivalent memory and processing power of a desktop machine; through to Personal Digital Assistants (PDAs), and programmable 'smartphones' with considerably less capacity. Other notable differences include styles of input (e.g. keyboard or keypad versus stylus or touchpad); display size, resolution and colour depth; and the presence or absence of non-visual output such as audio or vibrational alerts [8].

II. AD-HOC MOBILE NETWORKS

Given the heterogeneity of devices available to participate in next generation collaborative applications, this paper addresses the task of implementing mobile collaboration in what has been referred to as an ad-hoc mobile network [5]. Ad-hoc mobile networks are limited group environments defined by the following characteristics:

- devices with low processing capacity and simple input and output

This work is part of the research program of the Australian Telecommunications Cooperative Research Centre (ATCRC) http://www.telecommunications.crc.org.au.

- unreliable, low bandwidth wireless peer-to-peer communication
- dynamically formed based on contextual necessity in order to solve a specific problem

The applications described in the previous section present realistic scenarios for ad-hoc mobile collaboration, and given the potentially high payoff of successfully deploying such applications, provide a significant rationale in their own right. However, it is also argued that providing support for the development of applications supporting real-time synchronous editing of complex artefacts in limited ad-hoc computing environments is a significant technical challenge. As such, the development of such frameworks or techniques will prove instrumental in expanding the reach of other collaborative groupware systems. Finally, even for systems that usually operate in an environment with significant capacity and fast reliable communication, the work described in this paper will provide a fallback mode of processing that can facilitate continuity under extreme or unusual circumstances.

This paper investigates support for collaboration beyond simple asynchronous message passing, such as the synchronous cooperative development and editing of complex artefacts such as diagrams. In presenting a scheme for consistency management for synchronous collaborative editing applications within mobile ad-hoc networks, the contribution of this paper is twofold. First, by exploring the effect of applying traditional real-time synchronous collaboration algorithms in an ad-hoc mobile network, a number of important issues are identified. Second, in response to these issues, a mechanism for dynamically reducing and raising the levels of collaboration support to suit constrained mobile devices in ad-hoc networks is proposed.

The remainder of this paper is organized as follows: Section III begins with a review of existing work in mobile collaboration and identifies issues for further study. Section IV continues with a discussion of replicated architectures and the implications of using them in mobile environments, while section V covers the consistency management algorithms necessitated by such use. Section VI describes a consistency management algorithm suitable for use in mobile environments whilst section VII looks at how the algorithm can be dynamically adapted for use on constrained devices. Section VIII introduces MOCORA, a prototype that implements the adaptive consistency management algorithm described in previous sections; and section IX concludes with a description of limitations and opportunities for further work.

III. LITERATURE REVIEW - COLLABORATIVE MOBILE APPLICATIONS

This section briefly reviews work that has been done in adapting collaborative applications for use on constrained devices within mobile environments. In doing so, this section not only acknowledges existing contributions but highlights opportunities for further work.

Buszko et al. [5] describe a framework (YCab) that provides for a number of groupware services, including chat, whiteboard and image viewing. YCab recognises the need to provide the smallest code footprint for mobile devices, and thus the number of these services provided can be restricted on particular devices (e.g. textual chat only on a limited device), with services potentially optimised for each device type. Similarly, the MOTION framework [9] is a centralized system that provides for limited heterogeneous client devices by creating “cut-down” or customized, lightweight versions of software components.

Many applications are based on the distribution of code across multiple devices. DISCIPLE [3] implements a range of collaborative applications in a replicated architecture, and distributes applications across multiple devices when a single client device is unable to support the full application. DACIA [10] also supports dynamic configuration, with each client supporting a basic engine that remains at each client site, while additional components can be moved to different processing locations. Although the ability to re-configure applications across available resources is a desirable and elegant solution when there are sufficient resources, other approaches need to be available alongside dynamic re-configuration when implementing the critical peer-to-peer scenarios described in section I.

Low bandwidth communication, disconnection, lost messages and severe latency characterize ad-hoc mobile networks. C/Webtop [4] addresses limited wireless bandwidth by broadcasting graphical commands rather than complete display information. The MOTION framework [9] deals with disconnection using a publish and subscribe mechanism where newly connected clients can access stored messages.

Apart from the processing and storage limitations, applications may have to accommodate heterogeneity of input and output services. While not specifically addressing mobile applications, Li and Li [11] addressed heterogeneous applications software in a groupware environment by proposing a component in their framework that is able to take events in one editor (eg MS Word) and convert it for display in another (e.g. GVim). DISCIPLE [3] supports device heterogeneity by providing events in the form of XML messages, and having clients present them appropriately depending on the capability of the client device. The system also maintains client profiles, so that client data can be presented according to device preferences and requirements. DACIA [10] aims to provide a flexible groupware framework that deals with heterogeneous devices by allowing software to be assembled from pre-built components to meet particular implementation environments.

While the systems described above have proposed or implemented various techniques for dealing with the constraints of operating within mobile environments, none

have proposed a mechanism for dynamically modifying the consistency management algorithms that are central to the effective execution of collaborative editing tools in a decentralised environment. As a result, the following sections discuss the implications of using a replicated architecture, with its accompanying consistency management algorithms, in an ad-hoc mobile environment. Following on from this examination, the paper presents the MOCORA framework, which provides techniques for dynamically adapting the consistency management policies of a replicated collaboration architecture, when running on constrained mobile devices within an ad-hoc mobile network.

IV. THE REPLICATED ARCHITECTURE - IMPLICATIONS FOR THE MOBILE ENVIRONMENT

Unlike a traditional client server approach, a replicated collaborative architecture [12] maintains a copy of the application software and a copy of the shared workspace at each user node. The basic workflow is that a participant initiates a local operation at their site, has this immediately reflected in their local copy of the workspace or document, and then broadcasts the operation (through a *SendEvent* message) to other remote participants. Each participant receives these remote operations via a *ReceiveEvent* function that integrates each operation into their respective local workspaces or document copies.

Applied to collaborative systems, a replicated architecture can potentially provide greater fault tolerance, better local performance and the easy addition of new nodes. However, maintaining consistency across different sites, and enforcing strict What You See Is What I See (WYSIWIS) editing across all participants, introduces processing and storage requirements that may be difficult to meet in a mobile environment.

More simplistic techniques such as turn taking and locking can enforce consistency with less processing and storage overheads, but restrict the ability of users to concurrently access copies. This serves to undermine the intended usability benefits [12] of the replicated approach by interfering with the free flow of information, and thereby reducing system responsiveness for time critical activities.

Despite these difficulties, the potential benefits of the replicated architecture are compelling for mobile usage where disconnection and communications latency are more prevalent than in applications built around wired networks and desktop devices. Consequently, the following section discusses consistency management algorithms and the implications of applying them on constrained devices in a mobile environment.

V. CONSISTENCY MANAGEMENT

The independence inherent in the replicated architecture, and the latency and intermittent connections characterizing mobile networks means that local copies will frequently diverge. The aim of the replication management software at each site is to manage this divergence consistently, while supporting a high level of concurrency.

A. Causal Ordering

Underpinning consistency in real-time collaborative systems is the establishment of an order of events. The consistency model proposed by Sun, Zhang and Chen [13] provides a conceptual framework for devising schemes and algorithms to address the management of consistency in collaborative editing systems. This framework addresses the causal ordering of operations, the convergence of copies, and the preservation of user intention in maintaining consistency. This framework has been applied to replicated architectures for text-based editors including dOPT [14], adOPTed [15], LICRA [16], GOT [17] and GOTO [18], and for graphical editors such as ORESTE [19] and GRACE [20].

Lamport [21] defines a causal ordering relation ($\rightarrow$) as follows:

Given two operations O_a and O_b, generated at sites i and j, then $O_a \rightarrow O_b$, if and only if (1) i = j and the generation of O_a happened before the generation of O_b , or (2) i <> j and the execution of O_a at site j happened before the generation of O_b at site j, or (3) there exists an operation O_x , such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$.

Causality is preserved for any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is executed before O_b at all sites. Existing algorithms such as dOPT [14], ORESTE [19] adOPTed [15], GOT [17] and GOTO [18] achieve causality preservation by the use of a state vector, which is sent with each operation from a sending site. The receiving site uses this state vector to establish whether all preceding operations have been received.

In a replicated architecture, operations are applied as soon as possible, even if they are out of order. When a preceding event arrives, it may necessitate an undo/redo sequence, in which applied out-of-order operations are reversed using an application specific undo operation.

Since wireless communication suffers from increased latency and lost messages due to network dropouts, support for handling out of order operations is highly desirable. However, because storing out-of-order operations and providing application-specific undo processing requires additional resources that are dependent on the period of disconnection, handling unordered messages in a mobile environment on constrained devices may not always be possible.

B. Conflict

Concurrent access in a replicated architecture can also lead to conflicting operations. A conflict occurs when an operation O_a generated at site i cannot be executed at site j (i<>j) without reversing the intention of a user at j or at some other site. The conflict relation on operations has been

defined [20][22] as follows:

Two operations O_a and O_b conflict with each other, expressed as $O_a \leftrightarrow O_b$, if and only if (1) $O_a || O_b$ (O_a and O_b are concurrent); and (2) Target Object (O_a) = Target Object (O_b); and (3) Attribute.Value (O_a) <> Attribute.Value (O_b).

One solution is to consistently select one operation across all sites, and whilst this is effective, it will reverse a user's intention at one or more sites, and may not be suitable for all applications. It may be interpreted as not being in the spirit of cooperative work in some domains [23]. Often conflict will be an indication of contentious points that need to be discussed and explicitly resolved. For example, the association properties between classes in software engineering has important consequences for long-term operation of systems, and these must be well discussed and understood in coming to a final design.

As with causal ordering, conflicts are likely to be more prevalent in ad hoc mobile networks due to the greater periods of disconnectivity, and the resultant delay between operations being applied locally and reaching other nodes in the network. Again, conflict detection and resolution requires additional storage and application specific code to detect conflict, and to remove it by explicitly reversing one of the conflicting operations.

VI. AN ALGORITHM FOR MOBILE CONSISTENCY MANAGEMENT

The core of any replicated architecture is the processing of remote events by a 'ReceiveEvent' algorithm. Such an algorithm must maintain a causal order of operations and detect and manage conflicting operations.

The algorithm in Figure 1 is based on that first described by Karsenty and Beaudouin-Lafon [19], and modified by Wong et al. [23] to include conflict management. This algorithm captures the essence of many others, and is able to handle most documents including those that contain diagrams with object dependency. Since this algorithm supports out of order operations and conflict resolution, it is a candidate for use in mobile peer-to-peer environments. As a result, the implications of using this algorithm in such an environment are further discussed below.

Execute (e) is an application dependent procedure that applies the operation e.O to the workspace. Execute (e) must take account of maintaining the syntactic and semantic correctness of the workspace.

In a mobile network where latency is more apparent, operations that change a property of an object may appear at some site, before the event that creates that object. The algorithm deals with this by creating a list of pending operations (P), and applying these once the object has been created.

OrderEvents (e) maintains causal order by undoing any operations that follow e.O in causal order, applying e, then re-doing those operations that (were undone and) follow e.O. Again, undo is specific to the domain. One mechanism

```
Procedure ReceiveEvent (e: Event)
   -- e.O is the remote operation,
   -- e.T is the operation timestamp
   -- e.Obj is the target object
   -- Ts is the highest timestamp of an event applied at site s.
   -- P is log of operations pending the creation of an object.
   -- L is log of operations already executed.
   -- C is log of conflicting events (operations).
Begin
   If e.O conflicts with existing state,  store e in C, exit.
   If e.Obj does not exist, store e in P and exit.
   If e.T > Ts    -- causal order is preserved.
      Execute (e)  -- apply the operation
      Store e in L (log of  events that have been applied).
      Remove any conflicting event in C, resolved by e.O.
      If e.O creates an object
         execute operations of any event in P that has been
         waiting for this object to be created.
      Else
         -- an out of order event has been applied, so
         --  undo out of order events,  execute the new event
         --  redo the previously undone events.
         OrderEvents (e)
         --  remove any already sequenced operations from L.
         ReduceLog (e).
End.
```

Figure 1 - Consistency management algorithms for use in ad-hoc mobile networks

to maintain efficiency, and thus reduce resource consumption in mobile environments, is to use domain specific knowledge to establish a partial order by identifying operations that commute, or that mask others [19].

ReduceLog (e) keeps logs small by removing all events that have been applied in causal order. Again, this is a particularly critical activity when there is limited storage capacity, as is the case with the current generation of mobile devices such as PDAs and especially mobile phones.

When conflict resolution is enabled, the log of executed operations (L) also serves as a basis for determining conflicting operations by comparing newly received remote operations to the local log of operations already executed. If used for this purpose, then the log cannot be as easily reduced, unless a limit is made on the number of stored operations. Doing so however means that operations performed over periods of disconnection that exceed the length of the operation log would be automatically undone and rechecked for conflict upon reconnection, thereby increasing complexity and reducing local responsiveness.

VII. IMPLEMENTING ADAPTIVE MOBILE COLLABORATION SERVICES

Ideally, developers should be able to build applications once and maintain a single codebase, whilst relying on framework or tool support to deploy and configure such

applications on heterogeneous target devices and computing environments. Presently however, even application development environments such as Java, which have been designed with portability in mind, require different development approaches for desktop devices and high end PDAs (J2SE) [24]; mobile phones and low end PDAs (J2ME) [25]; and server-side components (J2EE) [26]. As such, current development tools require that applications developers design their applications with target machine capabilities in mind, and thus in the worst case maintain different code bases, or at best, different branches of a single source code tree.

In response to this difficulty, Ryan and Perry [27] have reviewed a number of methods for adapting code to different environments and proposed a framework [28] for the dynamic adaptation of mobile applications using declarative deployment descriptors and mobile objects. Although this work is motivated largely by the desire to exploit underutilized resources, it provides a useful background for customising collaboration software for execution on machines with diverse and sometimes limited capacity.

VIII. MOCORA - MOBILE COLLABORATION IN REPLICATED ARCHITECTURE

A. Application Adaptation

Building adaptation support into applications requires a study of the different components of a collaborative application. The main software components of a shared editor can be broken down into three main categories: 1) Domain specific workflows and business logic; 2) Input operations and presentation logic for the workspace containing the replicated document and its constituent artefacts; 3) Communication and collaboration support services including consistency management.

The domain specific workflows and business logic are generally fixed and can be designed independently of the target environment.

Regarding presentation, for the types of applications listed in Section I, the workspace can be constrained to some multiple of the display space. However, to achieve high usability, the workspace user interface should be explicitly customized to suit specific user environments.

In terms of the communication and collaboration support, the most non-deterministic, and hence the most troublesome aspect for constrained devices, is the size of the logs required to support consistency in the face of out of order operations and conflicts.

For example, the consistency management algorithm described in section V needs storage and software to deal with the management of conflicts and the ordering of operations. In theory, the pending (P), executed (L) and conflicting event (or replica) logs (C) used in the algorithm are unbounded. Mobile users may be disconnected and individual sites can be required to maintain out order operations and unresolved conflicts over relatively long periods. By reducing the level of support for consistency management, these logs and the procedures to support them can be eliminated. While this may reduce the quality of collaboration, it will allow the group to continue in situations where a less flexible algorithm would fail due to lack of resources.

In order to address the difficulty of supporting consistency management on constrained devices in mobile ad-hoc networks, MOCORA proposes three levels of support for consistency management:

- **UNORDERED_WITH_CONFLICTS:** Supports conflict handling by participants and unordered events. Requires 1) Pending log; 2) Executed Event log (L) and associated undo/redo methods; 3) Conflict log (C) and associated conflict detection and removal methods.
- **UNORDERED:** Implicitly resolves conflicts and handles unordered events based on pre-determined rules. Does not need conflict handling logs or methods.
- **ORDERED:** Handles only ordered events, and needs no logs.

UNORDERED_WITH_CONFLICTS provides the highest quality of support, but requires the most resources, and **ORDERED** provides the least service and the lowest use of resources.

Figure 2 shows the main components of the proposed MOCORA framework. The part bounded by the bold line (Receive Event, Application, Shared Copy and the Group Capability Manager) represents the core services needed to support remote operation processing in a collaborative environment. The Group Capability Manager tracks and enforces the highest supportable level of consistency management by monitoring local device capacity and initiating changes throughout the group if necessary. It is also responsible for responding to requests for consistency management change from other nodes, as described in more detail in the following section. The components outside the bold line (Conflict Manager, Order Manager and the logs) represent services and storage that can be removed from the mobile client.

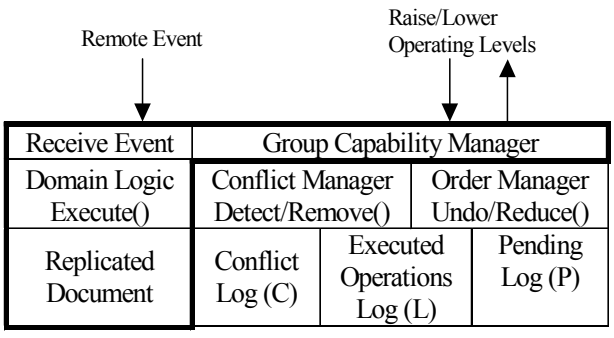

Figure 2 - MOCORA remote event processing engine.

B. Switching Levels of Collaboration Support

The level of consistency management support can be specified initially, for example by lowest common denominator, and be changed as a group session develops and the computational environment changes. The Group Capability Manager assesses available resources as each event arrives, and when it detects that the local device may not be able to continue supporting the current consistency management mode, it triggers a change in consistency management support throughout the whole group. Conversely, when a site becomes able to support a higher level of consistency management, it may initiate a move to raise the level of support for the whole group. These two scenarios are discussed further in the following two subsections.

1) Lowering the Level of Consistency Management Support.

Any site can unilaterally decide to move to a lower level of operation by sending the DECREASE_LEVEL message to all other participating sites, and all sites must match this mode. That is, when a site anticipates that it will not be able to operate at its current level, it moves into a lower level and instructs all other participating sites to move to this new level.

Moving to the **UNORDERED** level of consistency management from **UNORDERED_WITH_CONFLICTS** will initiate processing that no longer supports explicit conflict resolution. It removes the need for the conflict log (C) and code for handling conflicts at a site. Conflicting operations will be resolved by consistently selecting one operation across all sites based on pre-determined rules. If conflicts can be detected from the workspace, then Log L can be reduced to contain only out of order events.

Moving to the **ORDERED** level of consistency management from the **UNORDERED** level will remove the need to store data and to use code that deals with out of order operations. This includes the pending operations log (P) and the log of unordered operations (L), and any code for handling undo/redo sequences. This creates a difficulty in that some sites may have already accepted and executed out-of-order operations, so that clients may have to continue to operate at the **UNORDERED** level until a sequence has been established. The **ORDERED** level of operation means that sites will only accept operations in order. This implies that processing will revert to turn taking, and introduces the overhead of lock or token management.

2) Raising the Level of Consistency Management Support.

Raising levels is easy to achieve at individual sites, as each site can simply initiate logs and appropriate processing. It is more difficult to achieve across the group. One site cannot unilaterally raise the level. All participating sites must be able to accept the new level, requiring the use of a two-phase protocol. One site requests a higher mode of operation by sending a RAISE_LEVEL message to all participating sites. A site will decide if it wishes to operate at a higher level, and if so, it will issue a READY_TO_RAISE_LEVEL message to all other sites. When any site receives this message from all others, it will begin operating at the higher level.

In theory, some of the application specific processing required to deal with conflicts and unordered operations may need to be loaded into clients, and this implies that one site can act as a file server. It also implies that application specific code, for dealing with out-of-order operations and conflicts, should only be removed from a device as a last resort. Once this code is lost from all sites, it is lost from the group, and the ability to operate at this higher level is lost.

IX. LIMITATIONS AND FURTHER WORK

The adaptive consistency management algorithm described above has been partially implemented in a J2ME based prototype known as MOCORA (MObile COllaboration in a Replicated Architecture). Although MOCORA provides a proof of concept implementation and shows that such an algorithm can be specified in the constrained Mobile Information Device Profile (MIDP) environment of J2ME [25], it remains to fully specify and implement the complete protocol before the system can be considered production ready.

For example, a fixed number of known participants has been assumed and thus the impact of participants joining and leaving the group needs to be further explored since the application scenarios described in Section I require dynamic formation of groups.

There is also scope for more work on the monitoring of resources and deciding when a level of consistency management support can no longer be provided. Work on load balancing and dynamic re-configuration can be applied with a number of different criteria currently being assessed using simulations in MOCORA.

In its current state, MOCORA deals only with text and diagramming, and does not address richer multimedia applications that bring with them further complexities such as time dependency [12]. Time dependency requires that the time intervals between events be preserved across copies. That is, each site must not only have the same sequence of operations, but they must have the same time intervals at all sites. Dynamic situations such as search and rescue and military operations may require data to be received from different locations in a continuous stream, which must then be presented consistently across multiple nodes in real time. For example, if each user is viewing a moving image, and an object is selected or “marked” by one user, it must appear in the same way to all users (i.e. a marker attached to, and moving with, a common moving object).

In most cases, mobile devices will be deployed in environments that include some devices that are more powerful than others, as well as access to dedicated servers. The techniques described in this paper could potentially be integrated into this type of environment, thereby allowing collaborative applications to take advantage of other techniques like dynamic adaptation and object mobility.

ACKNOWLEDGEMENT

We wish to acknowledge Achyuthan Kumar for his work in implementing MOCORA.

REFERENCES

1. **Moyer, S. and A. Umar, *The Impact of Network Convergence on Telecommunications Software.* IEEE Comunications, 2001: p. 78-84.**
2. **Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. 1999, Reading, Massachusetts: Addison-Wesley.**
3. **Marsic, I. *An Architecture for Heterogeneous Groupware Applications*. in *Proceedings of 23rd International Conference on Software Engineering*. 2001: ISCE.**
4. **Bergenti, F., A. Poggi, and M. Somacher, *A Collaborative Platform for Fixed and Mobile Networks.* Communications of the ACM, 2002. 45(11): p. 39-44.**
5. **Buszko, D., W.H. Lee, and A. Helal. *Decentralized Ad-hoc Groupware API and Framework for Mobile Collaboration*. in *Proceedings of Group '01*. 2001. Boulder, Colorado.**
6. **Harr, R. *Exploring the Concept of Group Interaction through Action in a Mobile Context*. in *Proceedings of DEXA 2002 LNCS 2453*. 2002: Springer-Verlag.**
7. **Esbjornsson, M. and M. Ostergren. *Hocman: supporting mobile group collaboration*. in *Proceedings of Conference on Human Factors and Computing Systems*. 2002. Minneapolis, USA.**
8. **Tandler, P. *Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices*. in *Proceedings of UbiComp 2001: Ubiquitous Computing*. 2001. Berlin: Springer-Verlag.**
9. **Kirda, E., et al. *A Service Architecture for Mobile Teamwork onsistency model and supporting schemes for real-time cooperative editing systems*. in *Proceedings of SEKE'02*. 2002. Ischia, Italy: ACM.**
10. **Litiu, R. and A. Prakash. *Developing adaptive groupware applications using a mobile component framework*. in *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*. 2000. Philadelphia, USA.**
11. **Li, D. and R. Li. *Transparent Sharing and Interoperation of Heterogeneous Single-User Applications*. in *Proceedings of CSCW'02*. 2002. New Orleans, USA.**
12. **Begole, J., M. Rosson, and C. Shaffer, *Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems.* PACM Transactions on Computer-human Interaction, 1999. 6(2): p. 95-132.**
13. **Sun, C., Y. Zhang, and D. Chen. *A consistency model and supporting schemes for real-time cooperative editing systems*. in *Proceedings of Australian Computer Science Conference*. 1996.**
14. **Ellis, C.A. and S.J. Gibbs. *Concurrency control in groupware systems*. in *Proceedings of ACM Conference on Computer-Supported Work*. 1999.**
15. **Ressel, M., D. Nitsche-Ruland, and R. Gunzenbauser. *An integrating transformation-oriented approach to concurrency control and undo in group editors*. in *Proceedings of ACM Conference on Computer-Supported Work*. 1996.**
16. **Kanawati, R., *LICRA: A replicated data management algorithm for distributed synchronous groupware applications.* Parallel Computing, 1997. 22: p. 1733-1746.**
17. **Sun, C., et al., *Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems.* ACM Transaction on Computer-human Interaction, 1998. 5(1): p. 63-108.**
18. **Sun, C. and C. Ellis. *Operational transformation in real-time group editors: issues, algorithms and achievements*. in *Proceedings of ACM Conference on Computer-Supported Work*. 1998.**
19. **Karsenty, A. and M. Beaudouin-Lafon. *An algorithm for distributed groupware applications*. in *Proceedings of 13th International Conference on Distributed Computing Systems*. 1993.**
20. **Sun, C. and D. Chen. *A multi-version approach to conflict resolution in distributed groupware systems*. in *Proceedings of International Conference on Distributed Computing Systems*. 2000.**
21. **Lamport, L., *Time, clocks and the ordering of events in distributed systems.* Communication of the ACM, 1978. 21: p. 558-565.**
22. **Xue, L., M. Orgun, and M. Zhang. *Editing any version at any time: a consistency maintenance mechanism in Internet-based collaborative environments*. in *Proceedings of Ninth International Conference on Parallel and Distributed Systems*. 2002. Taiwan, ROC.**
23. **Wong, F., J. McGovern, and G. Fernandez. *Managing consistency and conflict in real-time collaborative software engineering*. in *Proceedings of the 3rd Argentine Symposium on Software Engineering*. 2003. Argentina.**
24. **Sun Microsystems, *URL: http://java.sun.com/j2se/*. 2003.**
25. **Sun Microsystems, *URL: http://java.sun.com/j2me/*. 2003.**
26. **Sun Microsystems, *URL: http://java.sun.com/j2ee/*. 2003.**
27. **Ryan, C. and S. Perry. *Client/Server Configuration in a Next Generation Internet Environment: End-User, Developer, and Service Provider Perspectives*. in *2003 Australian Telecommunications, Networks and Applications Conference (ATNAC)*. 2003. Melbourne, Australia.**
28. **Ryan, C. and S. Perry. *MobJeX: A Declaratively Configurable Java Based Framework for Resource Aware Object Mobility*. in *On The Move Federated Conferences (OTM '03) Workshops*. 2003. Catania, Sicily: Springer-Verlag.**