2011 10th International Symposium on Parallel and Distributed Computing

DeSCal — Decentralized Shared Calendar for P2P and Ad-Hoc Networks

Jagdish Prasad Achara and Abdessamad Imine
Nancy University and INRIA Nancy-Grand Est, France
Email: {Jagdish.Achara, Abdessamad.Imine}@inria.fr

Michael Rusinowitch
INRIA Nancy-Grand Est, France
Email: Michael.Rusinowitch@inria.fr

***Abstract*—This paper describes the design and implementation of a Decentralized Shared Calendar (abbreviated as DeSCal), a distributed application which provides users a decentralized infrastructure to share their calendar events with selected users in a dynamic group. Although being a distributed application, DeSCal is as responsive as a personal calendar. It achieves this high responsiveness by keeping a local copy of the shared calendar at each participating user. Consistency of these replicated copies of the shared calendar is carried out in a decentralized fashion using Operational Transformation (OT) approach. OT allows users to concurrently modify the shared calendar and exchange their updates in any order since it ensures the convergence of copies of the shared calendar in all cases. To prevent unauthorized access by illegal users, DeSCal is endowed with access control mechanism on the local copy of the shared calendar. It employs a flexible access control model based on replicating the access data-structure at each user site to overcome the latency problem. Also, access control on the shared calendar events is dynamic *i.e.* users are able to change the access rights on their shared calendar events at any point of time after the creation of an event. In short, DeSCal is totally decentralized, scalable and self-configurable *i.e.*, no need of a third party to manage it.**

***Keywords*-Consistency; Access Control; Decentralized; Scalability, Concurrency; Responsiveness;**

I. Introduction

Both personal as well as shared calendar provide a user the functionality to keep track of his important events. However, shared calendar simplifies the task of sharing an event information with other persons involved, which otherwise, has to be done through some other means like phone, email or personally meeting these persons. Further, it may not always be appropriate for a user to share each event in the calendar with other users in the group. A user may have some personal events which he would like to share only with some selected user(s) in the group, not to everyone. To deal with this aspect of shared calendar, there must be some mechanism for access control on calendar events. Here, sharing of an event may correspond to allow other users for any subset of <Read, Delete & Edit> operations on calendar events.

In order to avoid a single point of failure, the shared calendar should not depend on a central entity. A centralized shared calendar will no more work if the central entity is down and all participating users will be affected by it instantly. A decentralized shared calendar provides far better fault tolerance as compared to its centralized counterpart. An attacker trying the decentralized shared calendar to stop functioning will only be able to do so if he can successfully perform Denial-of-Service (DoS) attack on all the users of the shared calendar. On the other hand, it will take less efforts by an attacker in case of centralized shared calendar as he has to hijack only central entity on which centralized shared calendar depends. Besides, shared calendars based on a third party central server like Google Calendar [1] prevents users to create a dynamic Ad-Hoc group as users always have to communicate through this central server. In the case of third party dependent centralized shared calendar, there is no direct communication between users and they need to have a constant connection to this third party central servers (e.g. an Internet connection to connect to Google Servers to use Google Calendar). Above all, using a third party for sharing calendar events between a group of users may also lead to sacrificing the confidentiality of these users' events which is possibly an important concern for a user.

Funded by ANR Streams project and ARC INRIA Access project.

A. Motivation

The motivation behind this work lies in the design and implementation of a decentralized shared calendar where a third party is not needed. This makes shared calendar more robust against possible attacks and users no more have to tolerate the fact of disclosing their events to a third party. Subsequently, such a shared calendar should enable users to form a group locally in an organization where they can share their events with each other. Being a user-interactive application, it should consider human factors like [2]:

1) *High responsiveness*: the shared calendar must be as responsive as a personal calendar *i.e.* users should have an illusion that they are alone while using the shared calendar.
2) *High concurrency*: any number of users should be able to concurrently modify this shared calendar.
3) *Consistency*: users must eventually be able to see a converged view of all replicated calendar copies.
4) *Scalability*: the shared calendar must be dynamic in the sense that users may join or leave the application at any point of time during its runtime.

B. Contributions

We present DeSCal, an initial framework to satisfy all the above stated requirements of such a decentralized shared

978-0-7695-4540-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ISPDC.2011.40

calendar. We keep a copy of the shared calendar at each user site to improve performance of DeSCal. Users can perform updates on his local copy of the shared calendar independently and then, these locally executed updates are transmitted to other users in the group. To deal with latency and dynamic access changes, we use an optimistic access control technique (inspired from [3]) in such a way that enforcement of authorizations is retroactive. Access control is the ability to authorize or deny the manipulation of information by someone. A data structure is used to store all access rights. This data structure is checked whenever the controlling-access is started. We store this access data structure at each user site because high responsiveness can be lost if every update must be authorized by some authorization coming from a distant central server. To achieve our goal, we have to balance the computing goals of collaboration and access control to this shared calendar information in a decentralized fashion. Indeed interaction in shared calendar is aimed at making calendar events available to all who need it, whereas access control seeks to ensure this availability only to users with proper authorization. Due to replication and arbitrary exchange of updates, consistency maintenance in a scalable and decentralized manner is a challenging task. The challenges to develop such an application are 1) to handle the concurrent operations on shared calendar and 2) and to provide a mechanism for access control on these calendar events, both in a decentralized fashion. To the best of our knowledge, this is the first effort towards developing a decentralized and scalable shared calendar which is best suited for Peer-to-Peer (P2P) and Ad-Hoc networks.

C. Outline of the paper

The rest of this paper is organized as follows. Section II presents a real world scenario pointing out a possible deployment of DeSCal and outlines some related work in the field of shared calendar. In Section III, we present our collaboration model. Section IV describes overall DeSCal system by presenting its design and next, detailing architecture of a user site. In Section V, we present our implementation of DeSCal system on iPhone OS [4]. Section VI avails the usefulness of DeSCal system by presenting various scenarios. Section VII concludes our work and sketches possible future work.

II. Use case scenario and Related Work

A. Use case scenario

We present one real world scenario to illustrate the usefulness of DeSCal: A research team in an organization usually consists of a scientific and administrative leader, other members of the team and an administrative assistant. All members of the team can run an instance of DeSCal to keep track of their personal events and also, to share some group events with others. They can hide their personal events by not sharing these events with others. For group events involving two or more persons, one can create the event and share it with other members who are concerned by it. If few members of the team are in a group meeting, others can know this by just having a look on DeSCal. In addition, it is more intuitive for a team leader to share some administrative events with *delete* and/or *edit* right with the administrative assistant of the team so that extra hassle of communication can be avoided to deal with administrative tasks.

At the same time, we agree with the fact that this can be easily done using a centralized shared calendar too like Google Calendar but members of this research team can find DeSCal more appropriate if any one or all of the following possible scenarios are true:

1) The central entity in the centralized shared calendar is not owned by the organization itself and also, this organization has no connectivity with the outside network of a third party on which centralized shared calendar depends. For example, this research team can't use Google Calendar if they don't have Internet connection as it is managed by a third party.
2) Members of the team want to keep their calendar events confidential *i.e.* they don't want to disclose their calendar events with this third party (for instance, Google in case they use Google Calendar) who provides shared calendar service.
3) A team member meets someone while going for lunch or for a coffee and he wants to share some events with this person. This can be done instantly without any overhead like registering for shared calendar service of a third party; just by allowing other person to join the group and sharing only some specific events which this team member wants to share.

B. Related work

Shared calendars are common now-a-days but none of them is decentralized and self-configurable to the best of our knowledge. Google Calendar is a shared calendar application by Google but it needs a constant connection to Google Servers by each user and a prior registration for its shared calendar service. This follows that it can't be used over local Wi-Fi networks or over blue-tooth in an organization where mobile users can leave or join the group in an Ad-Hoc manner. This gives an edge to DeSCal over centralized shared calendars as users can use mobile version of DeSCal and join/leave the group in an Ad-Hoc manner. There exists mobile version of Google Calendar which is made for small screen and also, mobile phones' built-in calendars can be synchronized with Google Calendar when users are away from their desk but the need to depend on a third party (*i.e.* a constant connection to Google Servers) never goes away. Zimbra platform calendar application [5] enables users to share their events with other users in a group but again, this is centralized and a server needs to be run before using this application. While using Zimbra platform calendar application, one can run his own server in an organization

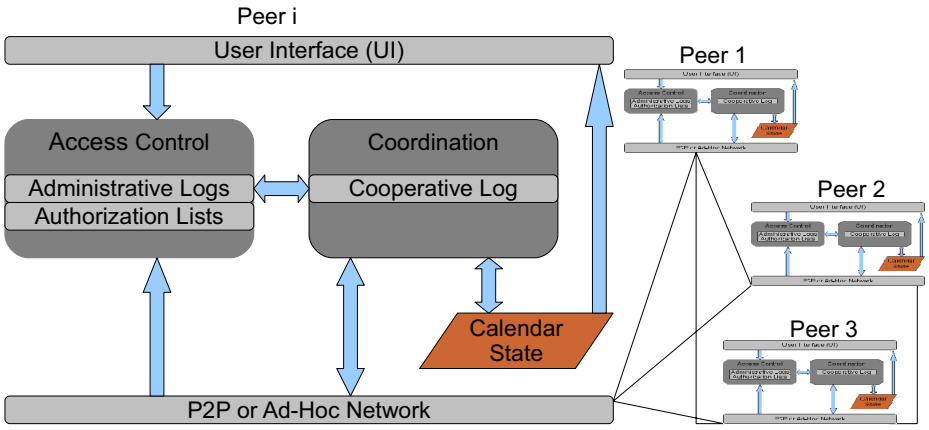

Figure 1. Design of DeSCal

without depending on a third party but centralized server prevents users to create Ad-Hoc group as users need prior knowledge of central server to run the application and also, this central server becomes a single point of failure for the shared calendar. Both Google Calendar and Zimbra platform calendar application uses CalDAV (Calendar extensions for Distributed Authoring and Versioning) [6] which is an Internet standard allowing a client to access scheduling information on a remote server.

III. Collaboration Model

In our collaboration model, we consider that a user maintains two copies: the *shared calendar* and its *access control policy*. A user in the group is the administrator for the events created by him and only an administrator can specify authorizations in his access control policy for the events administered by him. Users can modify an event in shared calendar with respect to the local access control policy of the administrator of that event. We define two types of operations: *Cooperative* and *Administrative* operations *i.e.* Operations concerning the state of the shared calendar and of the access control policy respectively. Our collaboration protocol proceeds as follows:

1) When a user manipulates an event in the local copy of the shared calendar by generating a cooperative operation, this operation will be granted or denied by only checking the local copy of the access control policy of the administrator of that particular event.
2) Once granted and executed, the local cooperative operations are then broad-casted to the other users. A user has to check whether the remote operations are authorized by his locally stored access control policy of this event's administrator before executing them.
3) When an administrator modifies his local access control policy by adding or removing authorizations, he sends these modifications *i.e.* administrative operations to the other users in order to update their local copies of the access control policy.
4) We assume that messages are sent via secure and reliable communication network, and users are identified and authenticated.

Here, we remind that copies of both the shared calendar and the access control policy are replicated at each user site as it is twofold beneficial: firstly, it ensures the availability of the shared calendar, and secondly, it allows for flexibility in access rights checking. However, this replication may create violation of access rights which may lead to fail meeting one of the most important requirements of DeSCal, the consistency of the replicated copies of the shared calendar. Indeed, the cooperative and administrative operations are performed in different orders on different copies of the shared calendar and the policy object. But thanks to the coordination [2] and access control [3] models used in DeSCal which ultimately ensures the consistency of the shared calendar at each participating node.

IV. DeSCal System

In this section, we present how DeSCal modules are organized in a running environment and then, we unravel the architecture of a user site describing how it handles local and remote updates on shared calendar.

A. Design

DeSCal has a generic design that can be deployed easily for decentralized architectures *e.g.* P2P. We describe the main building blocks of DeSCal. The design of DeSCal (Fig. 1) is composed of four well-separated conceptual modules: Coordination, Access Control, P2P/Ad-Hoc Network and User Interface.

1) Coordination Module: As DeSCal relies on decentralized architecture where no central server is required to be permanently on-line, it keeps a copy of the calendar at each participating node. Therefore, we need a mechanism which

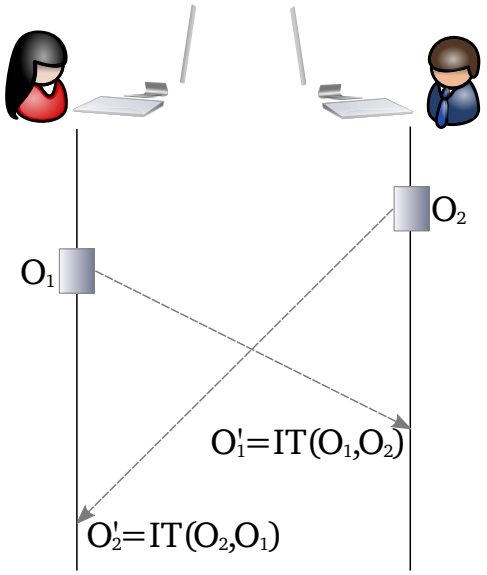

Figure 2. Operational Transformation (OT)

can handle the coordination of concurrent updates on this shared calendar by different users in a decentralized fashion. This mechanism should also handle the scalability feature of DeSCal where a user can join or leave the application at any point of time. We follow the coordination model proposed in [2]. This model ensures the same converged, consistent copy of the calendar at each node in all cases. It is based on Operational Transformation (OT) approach [7] .

This approach has been proposed in [8] for consistency maintenance in collaborative editors. It is an optimistic replication technique which allows many users to concurrently modify the shared data and next, to synchronize their divergent replicas in order to obtain the same data. It is considered as the efficient and safe method for ensuring consistency in a decentralized way without a need of any global order of updates. In general, it consists of application-dependent transformation algorithm, called IT, such that for every possible pair of concurrent updates, the application programmer has to specify how to integrate these updates regardless of reception order [2]. In Fig. 2, two users execute operations O_1 and O_2 on their local copy of the shared calendar (O_2 before O_1) and then, send it to other user. Operations O_1 and O_2 are concurrent in the sense that they are executed locally at their corresponding sites but their effect has not yet been seen on other sites. Upon reception of these operations at other user site, the received operation has to be transformed with respect to other in order to include the effect of other operation. One has to note here that reception of these locally executed updates at other user site is not guaranteed to be in the same order as they are executed locally at their corresponding local sites because of network latency variation, application execution speed depending on computer resources etc. Google Wave [9], Google Docs and many collaborative applications such as Joint Emacs [10], CoWord [11], CoPowerPoint [11] relies on OT approach for consistency maintenance. OT also has been proposed as a consistency model for replicated mobile computing [12].

There exists a variety of coordination models based on OT approach but we follow [2] because it is scalable, decentralized and fulfills all the requirements of a shared calendar for P2P and Ad-Hoc networks.

The coordination module directly interacts with the local copy of the calendar and is responsible for maintaining its consistency. It keeps track of both local and remote calendar update requests by storing them in a log called *cooperative log*. This log helps in maintaining the consistency of the shared calendar. In general, the collaboration is performed as follows: each user's updates are locally executed in nonblocking manner and then are propagated to other user sites in order to be executed on their local copies of the shared calendar. For more in-depth detail on coordination module for consistency maintenance, refer to [2].

2) Access Control Module: The whole copy of the shared calendar is kept at each user site to improve availability of data. However, we need to control the access to this local copy because users generally don't want to share all of their events with everyone in the group. We need to provide some access control mechanism to control access to these events so that a user is able to access the events for which he is authorized. Also, we remind that this access control mechanism on shared calendar events should be dynamic, decentralized and scalable as per requirements of DeSCal. Controlling access in such an environment is a challenging task as it needs dynamic access changes and low latency access to the shared calendar.

In addition, the requirements of DeSCal include high responsiveness of local updates. But, when adding a centralized access control layer, high responsiveness may be lost because every update must be granted by some authorization coming from a distant user (as a central server). So, we can't use a centralized access control model for DeSCal. To satisfy all requirements of DeSCal, we follow the decentralized access control model described in [3] where the shared calendar and its authorization policies are replicated at the local memory of each user. Thus, a user owns two copies: the *shared calendar* and the *policy*. It is clear that this replication enables users to gain performance since when they want to manipulate the shared calendar, this manipulation will be granted or denied by controlling only the local copy of the policy.

However, the access control model described in [3] doesn't satisfy all the requirements of DeSCal. The model proposed in [3] is single-administrator whereas in DeSCal, each user is the administrator of the events created by him. The single administrator model, proposed in [3], keeps a copy of the *authorization list* or *policy* and an *administrative*

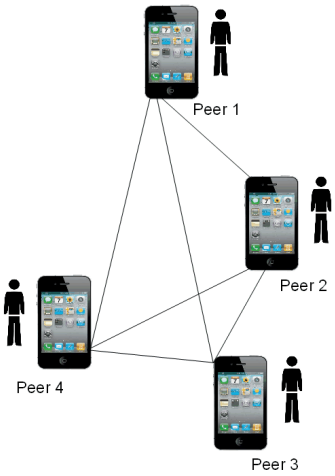

Figure 3. Peer-to-Peer Network of iPhone Users

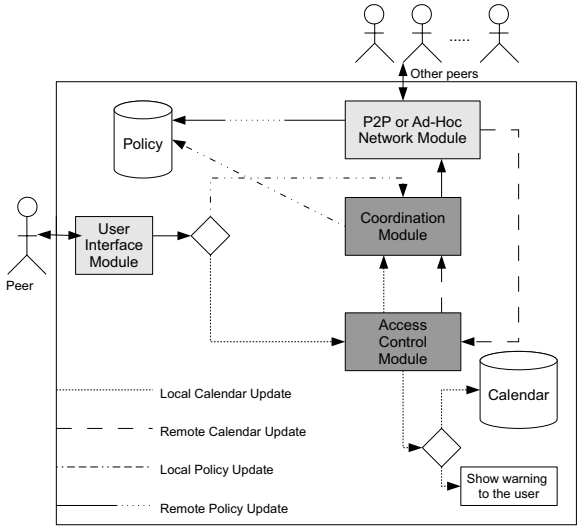

Figure 4. Architecture of a user site of DeSCal

log containing all administrative requests at each user site. We extend this model to make it a multi-administrator model where each user has his own 1) *authorization list* for the events created by him and 2) *administrative log* to store his administrative requests. Eventually, this requires each user to keep a copy of the policy and administrative log of all other users in the group.

The actions taken by a user on the shared calendar through user interface has to be passed through access control module and if authorized by this module, they are passed to coordination module to deal with consistency issues, which in turn, changes the state of the shared calendar. The updates on shared calendar and policy are applied in different order at different user sites. The absence of safe coordination between these different updates may cause security holes (*i.e.* permitting illegal updates or rejecting legal updates on the shared calendar). But the model proposed in [3] which is inspired by the optimistic security concept introduced in [13], relies on an optimistic approach that tolerates momentary violation of access rights but then ensures the copies to be restored in valid states with respect to the stabilized policy. For detailed description of this access control model, see [3].

3) P2P/Ad-Hoc Network Module: The role of this module is to maintain a local knowledge of the network infrastructure. It is the responsibility of this layer to provide Peer-to-Peer distributed architecture services to DeSCal for any kind of network like Wireless Ad-Hoc networks, Short range communication (e.g. blue-tooth), LAN, Internet or Managed infrastructure Wireless LAN. To implement this module, one can use an existing library capable of discovering network nodes independently *i.e.* without a need of a central entity (JXTA [14] is a good example of such a library) or can write his own library for this task.

The current implementation of DeSCal on iPhone OS uses Apple's Bonjour service [15] for discovering users in a local area network. This limits the use of current iPhone OS implementation of DeSCal in a local area network but it discovers both Wi-Fi and Blue-tooth users. Users communicate directly with each other and they have a total knowledge of the network infrastructure (Fig. 3). This allows users to create or join a group in an Ad-Hoc manner. All users are equally privileged in the system and a malicious user can't affect the application's execution for other users.

4) User Interface Module: It should be designed in a way that simplifies the use and hides the complexity of DeSCal to the user. It enables users to take actions on shared calendar. However, this module can't directly change the state of the shared calendar without interacting with the access control module. An action taken by the user on this module has to pass through access control module. If this action is authorized by access control module, it is passed to coordination module to change the state of the shared calendar and to deal with consistency issues.

B. Architecture

In this section, we present how a user handles the local updates generated by himself and remote updates by other users in the group received through P2P/Ad-Hoc Network module (Fig. 4). Both local and remote updates can be of two types: *Policy Update* or *Calendar Update*. Policy updates are generated if a user changes his policy whereas calendar updates are a result of various actions taken on the shared calendar. The various actions taken by the users can be inserting a new event and deleting/editing an existing event.

If the local update is a *Calendar Update*, it is passed immediately to access control module which checks whether the user is authorized to take this action on the shared calendar or not. Each user is authorized to insert an event in the calendar, so, a request to insert a new event in the calendar by a user will always be authorized by the access control module. Moreover, access control module will allow a user to take all actions on an event if it is created by him as, by default, he is the administrator of this event. However, for the event created by other users, access control module checks the policy of the corresponding administrator of that event and allows/denies the user to take that particular action on shared calendar accordingly. In our current implementation of DeSCal on iPhone OS, if a user is not authorized to take an action on shared calendar, a warning is shown to this user stating this fact and no change is made to the calendar state. As opposed to this, if a user is authorized to take an action, this local update is executed immediately on local copy of the shared calendar and then, it is forwarded to coordination model which deals with consistency issues. After processed by coordination module, this locally executed operation is broad-casted to all other users in the network and somehow, our coordination module manages to have the same copy of shared calendar at each user site with respect to the stable access control policy.

A local *Policy Update* is sent to coordination module which in turn, changes the local copy of the access control policy. Afterwards, this change in local copy of access control policy is broad-casted to other users to update their local copy of the access control policy.

A user can also receive these two types of updates from other users through P2P/Ad-Hoc network module. If the remote update is *Policy Update*, the local copy of the policy of that user is changed. Additionally, we store this administrative request in local administrative log of that user as a cooperative request can be undone if this remote administrative request is restrictive.

Lastly, if the remote update is *Calendar Update*, access control layer checks whether this operation is authorized or not and if authorized, sends it to coordination module for some further processing to deal with consistency issues. Moreover, if the receiver is the administrator of that event, the administrator generates and sends a special type of *admin* request: *Validate* to validate this calendar update to all users. This *Validate* administrative request is just to notify other users that the operation is granted by the local copy of the access control policy of the administrator and calendar updates can't be undone anymore.

V. Implementation on iPhone OS

Our implementation of DeSCal system on iPhone OS enables a user to share his events with other users to whom he wants to share. Each user has to manage his policy to control the access for other users on the events created by him. The interaction between the layers is limited and performed in such a way that it helps in making DeSCal more secure and robust. We explain here few key entities used in our implementation around which the whole story of DeSCal implementation revolves:

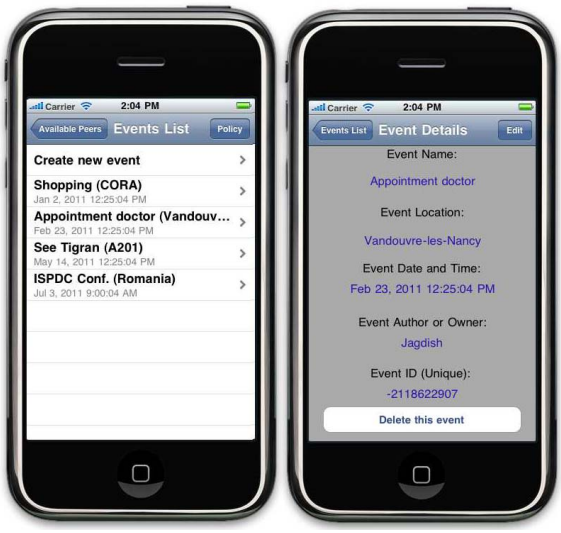

Figure 5. Calendar and Event Detail View in iPhone OS implementation of DeSCal

A. Key Entities

1) Calendar: It is list abstract type where each element of this list is an event. Events in this list are sorted in increasing order of date and time of the event.

2) Event: An event is an implementation based entity which may have variable user based attributes. In our implementation, an event has three attributes to be entered by the user: 1. *Event Title* 2. *Event Location* and 3. *Date and Time of the event*. In addition, our implementation stores a unique *event id* corresponding to each event to identify an event uniquely since other attributes can be same for two events. Furthermore, we need to have one more attribute to store the creator of the event. In DeSCal, an event is the smallest element for our coordination and access control modules.

Fig. 5 contains two snapshots of our iPhone OS implementation of DeSCal (Calendar view and Event Detail view from left to right).

3) Rule. : A rule is a quadruple <Peer(s), Event(s), Right(s), Permission> where Peer(s) can be any subset of all available users except the user who is inserting the rule; Event(s) can be any subset of all events created by that particular user; Right(s) can be any subset of set <Read, Delete, Edit>; Permission can be either attribution or revocation of rights.

To insert a new rule in the policy, our implementation allows a user to select the users from the list of available

Figure 6. Selection of various attributes to insert a new rule in policy in iPhone OS implementation

users, events from the list of events (only the events created by that user will be shown/available to select in this view), rights which (s)he wants to attribute or revoke and finally, Right Attribution/Revocation depending on whether (s)he wants to revoke or attribute these rights on these selected events for these selected users.

For demonstration, let's take an example where a user wants to give '*Delete*' and '*Edit*' right to a user named 'Michael' for his event: <Event 1> where Event 1 = {Title:Appointment with Doctor Location:Vandouvre-les-Nancy Date & Time:13 November 2008 02:00AM}. So, in this case, we will have to select 'Michael' from the list of users, 'Event 1' from the list of events, '*Delete*' & '*Edit*' from the list of rights(*Read, Delete & Edit*) and '*Right Attribution*' from the list of permissions(*Right Attribution & Right Revocation*). See Fig. 6 to have a look on how this rule selection is performed in iPhone OS implementation of DeSCal.

4) Policy: Policy is an indexed list of rules. When a user wants to delete/edit an event, the application needs to check whether the user is allowed to do that action or not. The application checks rules one by one in policy (indexed list of rules), starting from the first rule and stopping when it reaches the first rule that matches its request (based on *first-match semantics*). If no matching rule is found, that action can't be taken on the shared calendar. Fig. 7 contains two snapshots where left one is Policy view of our iPhone OS implementation of DeSCal.

B. Working

At start-up, application prompts a user to enter his/her name and then, shows the list of other available users (Fig. 7 right snapshot) participating in the shared calendar.

Figure 7. Policy and Available Peers view in iPhone OS implementation of DeSCal

Depicted below are three major actions a user can take in our DeSCal implementation:

1) Inserting a new Event: Our implementation allows each user to insert a new event in the calendar. The actual position of the insertion of the event in the calendar is determined by the application itself depending on event's date and time. Events are stored and displayed in increasing order of their date and time. Each event in the calendar has its unique event id, thus by, allowing two or more events to have the same attributes.

2) Managing Policy: A user can insert a rule anywhere in the policy and can delete any of the existing rules from the policy. The position of rule has importance in the policy

because application starts checking a request from the first (at the top) rule in the policy.

3) Deleting/Editing an Event: A user can always delete/edit the events created by him as he is the administrator of the event and can also delete/edit the events created by other users if he is allowed to do so.

VI. Various Scenarios demonstrating application's novelty

Using following three scenarios, we will demonstrate all the novel features of DeSCal.

A. Scenario One

Any number of users can simultaneously use this shared calendar, but the application will eventually have a single converged, consistent copy of the shared calendar at each user site. All users are allowed to insert a new event. By default, this new event is the user's private event and it will not be shared with other users in the group. However, at any point of time, this user can go to his policy view to share his event to all or a subset of users. Our application enables a user to revoke the rights attributed previously to some other user(s) as a user can change his mind later and he may decide to revoke the right for some user(s). Moreover, he may like to share his event to a newly arrived user. In brief, DeSCal provides high concurrency to users and access control on calendar events is dynamic as it can be changed at any point of time during application lifetime.

B. Scenario Two

Let's say, an event e_1 is created in the shared calendar by user u_1 and he allows user u_2 to delete e_1. Later, suppose, u_1 (administrator) of the event e_1 revokes the delete right for u_2 and at the same time, u_2 deletes e_1. These two operations are concurrent because they are locally executed at their respective sites but the effect of these operations are not seen at other sites. When u_1 will receive the remote delete operation, the delete operation will be rejected according to u_1's local copy of policy whereas e_1 is already deleted at user site u_2. This leads to inconsistent state of the calendar for a while. Since our approach is optimistic, we may tolerate momentary violation of access rights but we ensure the copies to be restored in valid states with respect to stable policy. When u_2 receives the revoke operation by u_1, our application detects this somehow that these two operations are concurrent. In cases where policy and calendar updates are concurrent, the priority is given to policy updates. So, the deletion of event e_1 will be undone at user site u_2. A calendar update can't be undone if the update is validated by the administrator of that event. By presenting this short scenario, we illustrate how DeSCal deals with the conflicts caused by change in policy and modification of events simultaneously. In short, DeSCal achieves consistency of shared calendar at each user site in all cases.

C. Scenario Three

When a new user joins the application, he retrieves the copy of the shared calendar, policies and administrative logs from a nearby user. After retrieval, this new user can participate in the same manner as all other existing users. If a user leaves the shared calendar, other users are not affected by it. Session is up and running if at least, one user is online.

VII. Conclusion and Future Work

We have described in this paper the design of a distributed application offering a decentralized and scalable environment for sharing calendar events with other users in the group. It is based on optimistic replication of both the shared calendar and the access control policies of each administrator in the shared calendar. As the design of DeSCal is scalable, it can be easily deployed on P2P and Ad-Hoc networks. DeSCal system is implemented on iPhone OS to test its deploy-ability.

In future work, we intend to improve security features for DeSCal like confidentiality of data, secure communication between users etc. as access control on local copy of the calendar is not sufficient to provide enough security against malicious users.

References

[1] "Google Calendar," http://www.google.com/googlecalendar/about.html.

[2] A. Imine, "Coordination model for real-time collaborative editors," in *Proceedings of the 11th International Conference on Coordination Models and Languages*, ser. COORDINATION '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 225–246.

[3] A. Imine, A. Cherif, and M. Rusinowitch, "A flexible access control model for distributed collaborative editors," in *Proceedings of the 6th VLDB Workshop on Secure Data Management*, ser. SDM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 89–106.

[4] "iPhone OS," http://en.wikipedia.org/wiki/IOS_%28Apple%29.

[5] "Zimbra Platform Calender Application," http://www.zimbra.com/products/calender-collaboration.html.

[6] "CalDAV," http://caldav.calconnect.org/.

[7] "Operational Transformation," http://en.wikipedia.org/wiki/Operational_transformation.

[8] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '89. New York, NY, USA: ACM, 1989, pp. 399–407.

[9] "Google Wave," http://www.waveprotocol.org/whitepapers/operational-transform.

[10] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," in *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, ser. CSCW '96. New York, NY, USA: ACM, 1996, pp. 288–297.

[11] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM Trans. Comput.-Hum. Interact.*, vol. 13, pp. 531–582, December 2006.

[12] R. Guerraoui and C. Hari, "On the consistency problem in mobile distributed computing," in *Proceedings of the second ACM international workshop on Principles of mobile computing*, ser. POMC '02. New York, NY, USA: ACM, 2002, pp. 51–57.

[13] D. Povey, "Optimistic security: a new access control paradigm," in *Proceedings of the 1999 workshop on New security paradigms*, ser. NSPW '99. New York, NY, USA: ACM, 2000, pp. 40–45.

[14] L. Gong, "Jxta: a network programming environment," *Internet Computing, IEEE*, vol. 5, no. 3, pp. 88 –95, may/jun 2001.

[15] "Bonjour," http://www.apple.com/support/bonjour/.