

Modeling, Simulation, and Practice of Floor Control for Synchronous and Ubiquitous Collaboration

Kangseok Kim

Community Grids Laboratory, Indiana University, Bloomington, IN, USA

Department of Computer Science, Indiana University, Bloomington, IN, USA

kakim@indiana.edu kskim@inha.ac.kr

Geoffrey C. Fox

Community Grids Laboratory, Indiana University, Bloomington, IN, USA

Department of Computer Science, Indiana University, Bloomington, IN, USA

School of Informatics, Indiana University, Bloomington, IN, USA

gcf@indiana.edu

Abstract: With the advances in a variety of software/hardware technologies and wireless networking, there is coming a need for ubiquitous collaboration which allows people to access information systems independent of their access device and their physical capabilities and to communicate with other people in anytime and anywhere. Current virtual conferencing systems lack support for ubiquitous collaboration. As the number of collaborators with a large number of disparate access devices increases, mechanisms for dealing with consistency in an application shared among collaborators will have to be considered in an unambiguous manner. In this paper we address issues related in building a framework for synchronous and ubiquitous collaboration. First, to make ubiquitous collaboration more promising, we briefly present a framework built on heterogeneous (wire, wireless) computing environment and a set of session protocols defined in XML to provide a generic solution for controlling sessions and participants' presences in collaboration. Second, to provide a solution for maintaining shared state consistency at application level, we present a floor control mechanism which coordinates activities occurred in synchronously cooperating applications being shared among collaborators. The mechanism with strict conflict avoidance and non-optimistic locking strategy allows all participants to have the same views and data at all times. Finally, we show modeling of XGSP-Floor control mechanism and formal verification to prove the correctness of the modeled control mechanism.

Key Words: Floor control, Ubiquitous collaboration, Synchronous collaboration, Mobile devices, Human-computer interaction

1. INTRODUCTION

Collaboration is about interaction among people and between people and resources. With the advances in a variety of software/hardware technologies and wireless networking, there is coming a need for ubiquitous collaboration and access which allows people to access information systems independent of their access device and their physical capabilities and to communicate with other people in anytime and anywhere. Also, with the maturity of evolving computing paradigms and collaborative applications, a workspace for working together is being expanded from locally collocated physical place to geographically dispersed virtual place. Mobile computing paradigm [34] made ubiquitous access possible with the integration of wireless communication technology in anytime and in anywhere. With grid computing paradigm [12, 20, 21] which is about sharing resources, resources are distributed into workspaces and shared among geographically dispersed collaborators. With pervasive computing paradigm [10, 32], it is becoming possible to make workspaces virtually suitable for collaborating users in the goal of all the time and everywhere instead of accommodating collaborating users to collocated workspace. We believe from Moore's law [31] and our development experience that the computing performance of mobile devices as well as desktop computers will continue to improve and networks' bandwidth will continue to increase. Thus the infrastructure improvements of software, hardware, and networking will make ubiquitous collaboration and access more prevalent and make the vision of Mark Weiser for 21st Century Computing [32] more promising as well in the future.

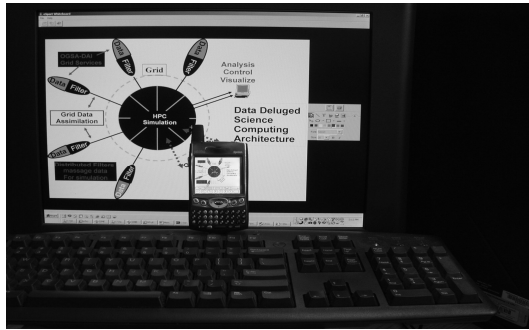


Fig. 1. A Screenshot of Collaboration between Desktop and Cell Phone

The following scenario illustrates the needs of ubiquitous collaboration and access, and motivates the research issues described in this paper. Researchers in Community Grids Lab (CGL) [5] at Indiana University often travel to attend offline real conference in a shared location. Students in CGL sometimes need to discuss with researchers. Researchers have to find a virtual conferencing system compatible with a conferencing system in CGL to discuss with students while traveling. Further, roaming researchers may have to find a place in which a compatible system is located. As this occurs, an integrated collaboration system, which combines heterogeneous virtual conferencing systems into a virtual conferencing system, will facilitate collaboration between the researchers and students. Virtual conferencing systems over Internet are rapidly increasing. Also, with increasing mobile devices, to integrate diverse mobile devices into a globally virtual conferencing system is becoming increasingly important. Current virtual conferencing systems lack support for ubiquitous collaboration and access. Fig. 1 shows an example screenshot of collaboration between desktop device and cell phone.

Students in CGL are going to have a session for their colleague's research presentation. Some students join the presentation session in a shared conference room of CGL and others join at remote locations by using CGL's conferencing collaboration tool – Global-MMCS system (Global Multimedia Collaboration System) [13, 16]. The presenter starts her presentation with the conferencing collaboration tool. During her presentation, she may use an application like shared whiteboard to discuss design issues of the research which she is doing on grid computing. In shared workspace with the application, people in offline shared real room see the same whiteboard canvas, while people in online virtual room see their own canvases. Each student in the online virtual room has their own canvas and a set of interfaces to the shared whiteboard application but they see the same results (or views) as others do. Her advisor, researchers, and colleagues in CGL want to make comments on her research by directly manipulating the shared application showing the same views among participants in her research presentation session. As participants in her research presentation session try to manipulate the shared application at the same time, she has to be able to provide the right to access the shared application for only one participant in the session at any time to ensure the consistency of the shared application state. The shared application, that requires mutual exclusions in real time, has to be assigned to only one participant who requests it under a set of well-defined rules. Participants in offline session can use the rules of etiquette or social protocols to gain the manipulation of the shared application in an order by the rules or protocols. However, participants in online session cannot use the etiquette rules or social protocols. Therefore, she will need some rules to substitute the etiquette rules and social protocols by defining the time and the way which a participant in collaboration gains access to the shared application.

This paper is organized as follows. Section 2 presents research issues. We discuss related works in Section 3. Section 4 briefly presents the architecture of collaboration framework and the implementation of it, and describes XML based General Session Protocol (XGSP). Section 5 presents a policy and a mechanism of XGSP-Floor integrated into our collaboration framework, and the functionality of an XGSP-Floor tool (floor control tool). Also, we present the conflict detection function and the non-optimistic locking mechanisms used in the XGSP-Floor for synchronous collaboration. Section 6 shows modeling of XGSP-Floor control mechanism. Then, we show formal verification to prove the correctness of the modeled control mechanism. Finally we conclude with a brief discussion of future work.

2. Problem statement

Conference collaboration systems typically provide a group of users with a set of well-defined interactions to access applications and resources, and communications among them. In such collaboration systems a group of users generally work sharing collaborative applications and resources in their workgroups (sessions). The cooperation on the resources shared among a group of users may hence produce new results on the shared resources. In traditional face-to-face offline session, participants generally follow rules of etiquette or social protocol when they interact with each other. For example, if all the participants try to draw on a shared whiteboard, then the conflicts which may result in inconsistent state can be solved by a moderator or social protocols. However, in online session or CSCW (Computer Supported Cooperative Work), the social protocols may not be able to be used for coordinating the interaction of participants since they are not collocated. For example, if all the participants try to send drawing events through a communication channel in a distributed collaboration system, then the conflicts are not able to be solved by the social protocols used in face-to-face offline session. Therefore, policies and mechanisms used in an offline session may need a mapping into those able to be used in an online session with user interfaces between participants and CSCW environment. When users perform concurrent activities on shared synchronous resources such as collaborative applications, floor control [7, 8] is necessary. Floor control is the problem of coordinating activities occurred in synchronously cooperating resources shared among participants in an online conference session. The floor control mitigates race conditions within online sessions on who is allowed to manipulate shared data or to send synchronous events. A set of well defined policies and mechanisms are needed for efficiently coordinating the use of resources in CSCW. The policies for floor control typically describe how participants in CSCW request resources, and how the resources are assigned and released when participants share a synchronous resource such as audio-video control event in conferencing, drawing events in shared whiteboard or moving events in chess game. Also, mechanisms including user interfaces (human-computer interaction) between participants and CSCW environment are needed to implement and enforce the policies. The floor control mechanisms have to be able to provide the floor on shared resource for only one participant in a synchronous online session at any time. No single floor control scheme may be appropriate for all collaboration applications. The simplest scheme is free-for-all (no floor control) for applications like text chat. Therefore floor control needs to provide significant flexibility ranging from free-for-all to application specific floor control mechanism for avoiding uncoordinated activities to shared collaboration applications.

3. Related works

In this section we examine existing floor control schemes for collaboration system. Dommel [8, 9] classified floor control schemes into two known paradigms, random-access (contention-based) and scheduled-access (token passing-based) floor controls. The random-access based floor control scheme includes sensing availability of a resource by users or system, or mediation by social protocols. The sensing floor scheme example is Activity Sensing Floor Control (ASFC [25]). The ASFC provides a mechanism based on sensing activities on a distributed shared resource. By sensing activities on the shared resource, decisions for floor control are autonomously made without the mediation (intervention) of moderator. If the requests are collided, they are backed off until the floor holding the resource is released. The example protocol schemes by mediation are Conference Control Channel Protocol (CCCP [30]) and floor control protocol built on MBone seminars [27]. The CCCP provides moderator-controlled interaction floor control in conference collaboration where a designated moderator gives access rights to a participant who wants to access a shared resource. The MBone videoconferencing system uses centralized moderator-controlled floor control mechanism in question board which is a tool to enable participants to ask questions in large-scale loosely-coupled MBone seminars (sessions). It also enforced the floor control mechanism using the conference bus mechanism developed at LBL (Lawrence Berkely Laboratory). The conference bus is a multicast mechanism which is used for communication between tools provided in a session. The scheduled-access based floor control scheme includes autonomous token passing interaction floor control scheme where the token is used to request, grant, deny, or release a floor. Boyd [2] classified the floor control schemes according to design dimensions such as degree of interaction and granularity of control for floor control policies in multi-user applications. He introduced an interactive and fine-grained policy with user interface for floor control, called fair dragging which can be used in multi-user applications. The policy means that a user has control of an object during only short-

term dragging, where dragging means pressing a button over an object which he wants to use and releasing the button when he relinquishes the use of the object. Greenberg [17, 18] classified the floor control schemes for turn-taking between participants with view-sharing applications. He discussed some floor control mechanisms implemented in the view-sharing applications and showed as an example that explicitly managing (or designing) turn-taking floor control with view-sharing (of full screen) applications in a windowed environment is difficult without disturbing the shared view to explicitly activate floor control from user's perspective because there is no room to display information about current state of the floor in the shared full screen. Also, GroupKit [18, 28] provides participants in collaboration with flexible floor control mechanisms – preemptive floor control scheme and ring-passing scheme. The preemptive floor control scheme means that a user can immediately grab the floor from the current floor holder. The ring-passing floor control scheme means that a user can seize the floor if the floor is free. Myers [4] created more comprehensive classification of floor control policies with three independent primitive dimensions: request, acquire, and release control. By combining the primitives, he showed the use of the floor control with puzzle control program in Pebbles project [3] which studies the use of PDAs (Personal Digital Assistants) simultaneously with a desktop PC.

4. Collaboration framework architecture and XML based General Session Protocol (XGSP)

Collaboration framework is a basic structure to hold consistent view or information of users' presence and sessions together, and to support diverse collaborative applications to collaborators joining in a conference at remote locations. It also has a capability that allows a user to join a conference using networked heterogeneous (wire, wireless) computing devices anytime and anywhere and to use collaborative applications in the conference. It is important to users joining a conference that it seems to be in offline real conference room even when using heterogeneous computing devices at remote locations. It is typical today and will be more typical in the future that all users can access information independent of their access devices and physical capabilities anytime and anywhere. To maintain consistent information of presences and sessions in a conference, we use a request (query) and response (dissemination) mechanism that requires a user to inquire queries (request event messages) to a chairperson node (conference chairperson) and a conference manager in order to engage in presence and various collaboration activities, and the chairperson node and conference manager to disseminate the queried information to all the participants through our messaging and service middleware – NaradaBrokering [33, 36, 37]. A set of protocols are defined in section 4.7 for maintaining consistent collaboration state information among participants in conference collaboration. As shown in Fig. 2, the collaboration framework is structured as three layers and six major components: control manager, session / membership control manager, access / floor control manager, policy manager, request and reply event message handlers, and communication channel. We describe the components in turn.

4.1 Control Manager

A control manager is an interface component located between sessions and managers in collaboration framework for providing conference management services such as presence, session, and access and floor control managements for participants in collaboration. Presence of participants, creation/destroy of sessions, and activation/deactivation of actions to access resources are serviced through this manager into each of control management services. The control manager also has factories for all kinds of applications, and hence can create new application instances and invoke, start, and destroy them.

4.2 Session and Membership Control Manager

This manager manages information about who is currently in the conference and has access to what applications, and which sessions are available in the conference. The session and membership control manager has a set of control logics that are used to manage presences of and connectivity among collaborating users in collaborating workgroups, and organize the workgroups. The control logics communicate through a set of predefined protocols (session control protocols) for streaming control messages to exchange presence information of collaborating users and state information of various collaborative sessions. The session control protocols account for policy, presence, session creation, initiation, teardown, and so on. To describe presences, connectivity, and states of sessions, XML is used as a protocol definition language of the session and

membership control. The XML based General Session Protocol (XGSP) is described in section 4.7 briefly and in more detail in [41].

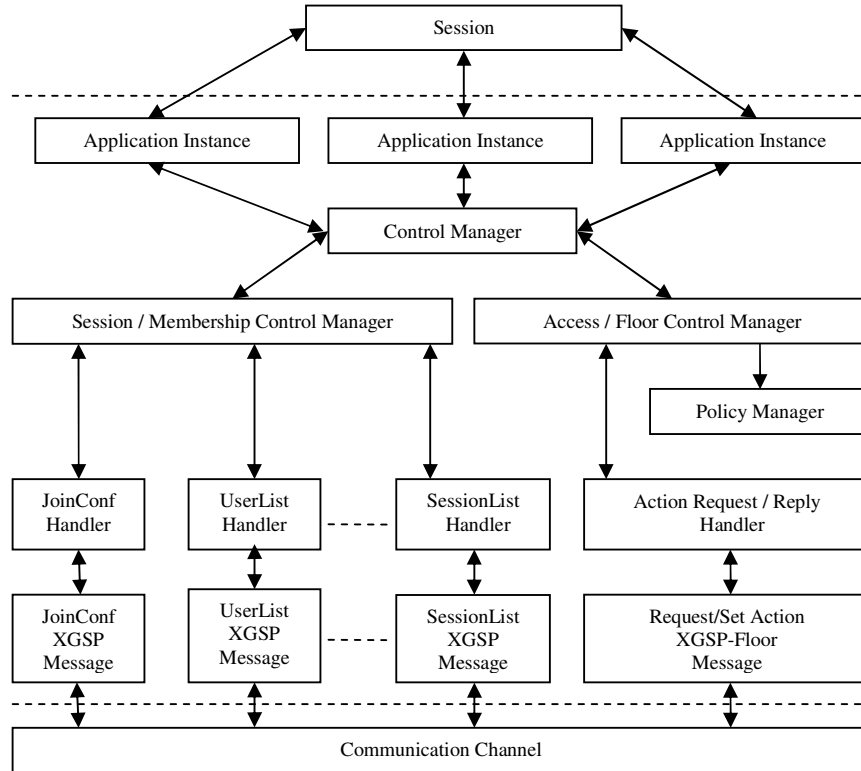


Fig. 2. Collaboration framework architecture consists of three layers (collaborative applications, managers, and communication service) and six major components.

4.3 Access and Floor Control Manager

The access and floor control manager component in the collaboration framework is responsible for handling accesses to collaborative applications through the request and reply event message handlers which are one of components in the framework. A user requests an access to use resources like collaborative applications to a chairperson or moderator through a request event message handler. The chairperson or moderator responds a decision (grant, deny, or queued) to the requesting user who wants to access resources through a reply event message handler. The chairperson or moderator also broadcasts the decision to make the change of access state to each resource globally visible to all the participants in a session. A GUI (Graphic User Interface) on the framework, which is used to display access state information for resources, is used to request accesses to resources. Within the access and floor control manager, policies are read from a file, a request is validated through a policy manager and one of classified access types is returned into the manager through an access type decision service. With the returned access type, a chairperson or moderator makes a decision and the decision is dispatched to the requesting user. Also the decision is broadcasted into each node to update the access state information for the resource. The XGSP Floor control mechanism (XGSP-Floor) is described in section 5 in more detail.

4.4 Policy Manager

Access and floor control policies are written in XML and put into the conference manager which resides on web server running on tomcat for globally consistent use. When a new user joins a conference, the conference manager pushes the policy into the node (or host) of the new user as a stream message, and the policy is stored in local policy store (a file) of the node during joining (connecting) in the conference. The policies describe which roles (users in them) in collaboration are allowed to perform which actions on which target applications. As a request event message for accessing applications arrives, the policy manager pulls the policy from the policy store. The

policy manager is responsible for validating the request event messages based on the access and floor control policies pulled from a local policy store.

4.5 Request and Reply Event Message Handlers

An event message handler is a subroutine that handles request and reply event messages. The control manager manages the associations between incoming and outgoing event messages with each of event message handlers. According to the associations, generated outgoing (request) event messages are first processed by the associated request event message handlers in each node (or host). Incoming (reply or response) event messages are also serviced by the associated reply/response event message handlers. The messages are sent to a broker (messaging and service middleware) via the communication channel shown in Fig. 2. The broker disseminates the messages to other participants connected to the collaborating workgroup.

4.6 Communication Channel

The communication channel is responsible for controlling interactions among participants and communications among collaborative applications. The channel uses topic-based publish-subscribe mechanism that defines a general API for group communication. The API for the topic-based publish-subscribe mechanism is used as an interface for group communication of sessions in a conference and between collaborative applications and a broker. In the topic-based publish-subscribe mechanism, the topic information contained within messages is used to route the messages from publisher to subscriber. The topic information has two kinds of naming schema: a name separated simply by slash("/") strings like /XGSP/Conference-ID/Application-Session-ID can be used and another naming schema can be described using a set of tag=value pairs, a set of properties associated with the message, verbose text, or XML. The messages containing topic information are sent to a broker through the communication channel. And the messages are disseminated through router nodes, referred to as brokers to subscribers which registered a subscription to the topic.

4.7 XML based General Session Protocol (XGSP)

Collaboration can be defined as interaction for cooperation on shared resources among people working at remote locations. The interaction in collaborative computing requires a simple and universal access means and mechanism for people to easily access information or to conveniently communicate with other people. Interactions and cooperation for collaboration can be generally provided through the unit of conference and sessions. A conference is composed of a set of sessions, where a session means online workgroup of collaborating users working with sharing various collaborative resources. A conference needs control logic to maintain state information among sessions and presence information among participants in a conference. The control logic is used to manage presences of and connectivity among collaborating users in the online workgroup (session), and organize the online workgroups (sessions or conference). The control logic needs a protocol for streaming control messages to exchange presence information of collaborating users and states of various collaborative sessions. To describe control logics of presences, connectivity, and sessions' states, we use XML as a protocol definition language of session control. The XML based General Session Protocol (XGSP) [41] is a protocol for streaming control messages written in XML to provide various collaboration sessions in a conference for users according to the presences and connectivity. The session control protocol account for policy, presence, session creation, initiation, teardown, and so on. The details of conference, session, and presence management protocol are described in [41].

5. XGSP Floor control (XGSP-Floor)

This section describes a floor control policy, a floor control mechanism (XGSP-Floor) implementing the floor control policy, and the functionality of a XGSP-Floor control tool to provide participants in collaboration with human-computer interaction for control of a floor, where the human-computer interaction means user interfaces between participants and CSCW environment. We also describe a conflict detection function and a non-optimistic locking mechanism used in the XGSP-Floor.

```

<XGSP-FloorPolicy>
  <ResourceAccesspolicy>
    <ResourceAccess>
      <RoleName>mobile-user</RoleName>
      <ApplicationRegistries>
        <ApplicationRegistry>
          <ApplicationID>wb</ApplicationID>
          <MainClass>cgl.myprofessor.whiteboard.Whiteboard</MainClass>
          <Actions>
            <Action>
              <ActionName>slave</ActionName>
              <Capabilities>read</Capabilities>
              <AccessType>released</AccessType>
            </Action>
            <Action>
              <ActionName>line</ActionName>
              <Capabilities>linedrawing</Capabilities>
              <AccessType>shared</AccessType>
            </Action>
            <Action>
              <ActionName>oval</ActionName>
              <Capabilities>ovaldrawing</Capabilities>
              <AccessType>exclusive</AccessType>
            </Action>
            <Action>
              <ActionName>pen</ActionName>
              <Capabilities>freedrawing</Capabilities>
              <AccessType>exclusive</AccessType>
            </Action>
          </Actions>
        </ApplicationRegistry>
      </ApplicationRegistries>
    </ResourceAccess>
  </ResourceAccesspolicy>
</XGSP-FloorPolicy>

```

Fig. 3. An example of XGSP-Floor Policy with the Role Name mobile-user and Application Name whiteboard (wb)

5.1 XGSP-Floor Policy

This section describes an XGSP-Floor policy (floor control policy) that defines how the participants in synchronous collaboration session request a floor for the use of a collaborative application, and how the floor for the use of the application is assigned and released when the participants share the synchronous collaboration application. An example XGSP-Floor policy written in XML is shown in Fig. 3. The element access type in the example policy describes whether a fine-grained action of an application can be shared among participants. If a fine-grained action is not able to be shared among participants, a floor control mechanism has to be able to provide a floor for the action on a shared application for only one participant in a synchronous collaboration session at a time. We define a set of predicate rules (policies) used as determinants in decision procedure of a moderator node in terms of the following three types of predicate statements (request, response, release) to provide a floor for only one participant at a time:

1. Participants in a synchronous collaboration session can request a floor for the use of a shared application in the session using the XGSP-Floor control tool described in section 5.3, or a moderator in the session can directly assign a floor to participants. This case, which is a mapping from offline request-response social protocol into online human-computer interaction, is for the shared whiteboard in our collaboration (moderator-mediated request-response interaction scheme). The text chat application does not need the floor request for free conversations among participants (no floor control scheme). The collaborative chess game application [39, 43, 44] uses different floor control scheme which alternates a floor in turn between the two players using the roles white-player and black-player in our collaboration role term (two-player turn-taking scheme or token-passing scheme).
2. When a participant requests a floor on an application being shared among participants, after the floor request is validated by the access and floor control decision service of a moderator node,

- If the floor is available, a moderator assigns the floor to the floor requester.
 Otherwise, the floor request is queued into a floor waiting queue or can be denied.
- When a current floor holder releases a floor control or after a prefixed amount of time, the floor is assigned to a requester waiting in a floor waiting queue in FIFO order or the floor can also be directly released from a moderator.

5.1.1 Collaboration Role in XGSP-Floor

Collaboration roles in XGSP-Floor are a representation to categorize collaborating users joining a conference session for collaboration. The roles are based on the users' privileges and devices' capabilities to manipulate protected shared collaborative applications. In this section we present how collaboration roles used in XGSP-Floor are represented. For the representation we use functional notion to show the relationship between roles and action privileges.

In role abstraction domain of the function we express the collaboration roles to be assigned to users joining sessions. In action representation domain of the function we express actions permitted to manipulate protected collaborative applications in sessions. The function representation is shown in Fig. 4. The definition of the collaboration actions depends on the type of applications. As an example we use shared whiteboard application for the definition of actions in Backus-Naur Form (BNF) below. In BNF we also define collaboration roles and actions as follows.

```

CollabApp ::= WB
CollabRole ::= Chairperson | Moderator | Non-mobile User | Mobile user
CollabAction ::= Master | Slave | Line | Rect | Oval | Pen | Eraser | Clear | Load | Move
  
```

5.1.2 Fine-grained Action (Interactive Smallest Major Event)

We define fine-grained actions in our collaborative application as the smallest interactive major events (semantic events [44]). For example, in the whiteboard application, drawing a line includes clicking, dragging, and releasing a mouse on the whiteboard canvas. For a user working alone with the whiteboard, user input events (low level events such as mouse click, drag, and release) can be interactive major events between the user and whiteboard application. For users working with others sharing the application, the smallest major event means "drawing a line" (semantic event) and the user input events will then be an event data (mouse click – the first point of the line and mouse release – the second point of the line). CGL built a shared SVG (Scalable Vector Graphics [35]) browser and a collaborative chess game application with SVG [39, 43, 44]. In the collaborative chess game application, the smallest major events are to click on an object, to move, and to release the object during moving the object. After the completion of each move (as the mouse is released), the semantic event (moving an object) is dispatched to another player as the smallest interactive major event. Then the user input events will be an event data for moving an object in the chess game affecting the chess board (view-sharing) of another player as well as observers. Therefore, the major events can be different according to the types of applications. The fine-grained action in our collaboration means an interactive smallest major event affecting the shared view (or result) among users in collaboration.

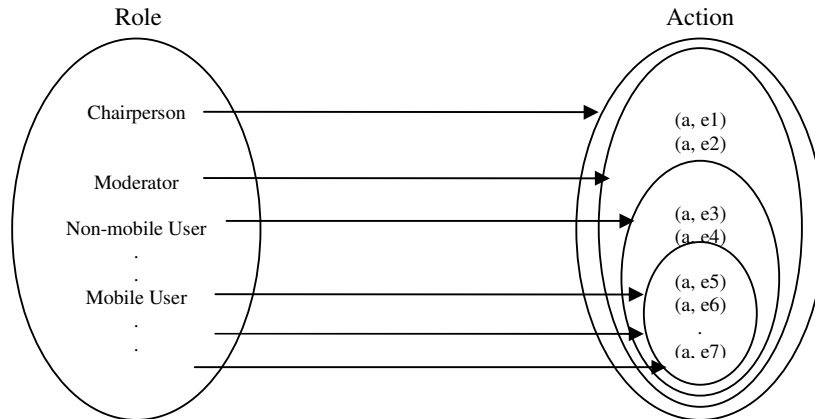


Fig. 4. A collaboration action is represented as a pair $(a, e) \in A \times E$, where $a \in A$ is an application and $e \in E$ is the authorized smallest major event defined by a , and A is a set of applications, E is a set of the smallest major events defined by an application in A .

5.2 XGSP-Floor Mechanism

A floor control mechanism (XGSP-Floor mechanism) is a means to implement the floor control policy described in the previous section. An XGSP-Floor mechanism regulates floors among all the participants in collaboration. In this section we present decision procedures implemented in a moderator node (which is a decision node to control accesses for applications in our collaboration system) to determine grant or deny of participants' floor requests to access applications in moderator-mediated interaction mechanism. The decision procedures follow the following five different types of stages. The broad view of the moderator-mediated interaction mechanism is depicted in Fig. 5. Also, we show a request-response interaction scheme between a moderator and a floor requester with human-computer interaction in Fig. 6.

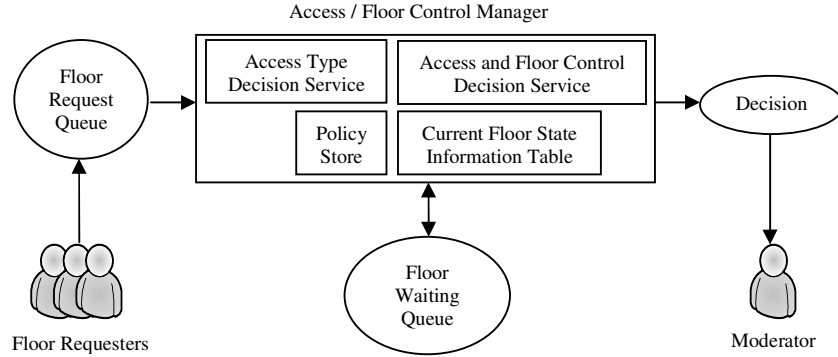


Fig. 5. Decision Procedure of XGSP-Floor Mechanism

5.2.1 Decision Procedures of XGSP-Floor Mechanism

First, a moderator node has a single queue for storing floor requests from participants. The queue is implemented in FIFO (First-In, First-Out) order for mitigating race conditions of floor requests to applications and thus enforces mutual exclusion among applications. The first request in the queue is validated by policy manager and is sent to the access type decision service located in the access and floor control manager of the moderator node. Then, the first request is removed from the queue. During the activity, new floor requests are stored in the floor request queue waiting for next service. Second, the access type decision service returns a classified access type value among Invalid, Implicit, Exclusive, Shared, or Released into the access and floor control decision service in access and floor control manager. Third, decision activities are behaved with the same type value returned from the access type decision service. The decision activities are also classified (or branched) into the same access type activities as the returned value mentioned in the second stage. Each decision activity returns one of decision values (grant, deny, or queued) to the moderator. Then, the moderator can make a decision according to the decision values. In this stage, we present a set of predicate rules used as determinants in decision procedures of a moderator node in terms of the following two types of predicate statements:

- 1 Determination of types classified to access applications.
 - a) If the role name, application ID, and request action of a floor requester is validated by a policy manager then an access type value (among Implicit, Exclusive, Shared, or Released) by access type decision service is returned into the access and floor control decision service. In the elements, the role name means the name of role assigned to participants in our collaboration system, the application ID means an application identifier existing in application registries of our collaboration system, and the request action means the name of a fine-grained action in which participants can manipulate applications.
 - i. If the return type is "Implicit", then the request is granted.
 - ii. If the return type is "Exclusive", then the request is granted or queued.
 - iii. If the return type is "Shared", then the request is granted or denied.
 - iv. If the return type is "Released", then the request is granted.
 - b) If one of the elements does not exist in policy, then a type "Invalid" is returned into a moderator and the request is denied.
- 2 Determination of whether an action in a request exists in current floor state information table, in other words, a request action conflicts with the action of current floor holder.

- i. If the return type from access type decision service is “Exclusive” and the request action exists in the floor state information table of a moderator node, then the request is queued. Otherwise, the request is granted.
- ii. If the return type is “Released” and a floor waiting queue is not empty, then the request is granted and the first request in the waiting queue is granted and removed from the queue.
- iii. If the return type is “Released” and a floor waiting queue is empty, then the request is granted.

Next stage is to update current floor state information table. To maintain consistent shared state at application level among collaborating participants, we need to maintain the current floor state information. This floor state information is updated to reflect an action in a request whenever the request is granted. Finally, all the requests stored in a floor waiting queue for the use of shared applications are serviced in prefixed amount of time to avoid starvation. The floor requests to shared applications are stored in a single queue which is implemented in FIFO order. The first request in the queue is serviced when the floor of current floor holder is released or after a prefixed appropriate amount of time. Then, the request is removed from the queue and the current floor state information table is updated with the removed request. Note that when the floor of current floor holder is released after an appropriate amount of time, the mechanism uses the acknowledgement (reply message) from the revoked node. The acknowledgement prevents the floor of current floor holder from assigned to another participant before the floor of the floor holder is revoked.

5.2.2 Request-Response Interaction Scheme between a Moderator and a Floor Requester with Human-Computer Interaction

The moderator in our collaboration system is responsible for passing floor control to and from participants in XGSP-Floor mechanism. The moderator grants a floor either by clicking on a button on pop-up window representing a participant’s request or by selecting an entry from the action list allowed for the participant displayed on a frame window invoked from a moderator node manager. The communication channel in a moderator node (on which a moderator resides) shown in Fig. 2 disseminates the floor to all the participants in a session through a broker including a decision (grant) for the floor requester (a participant who wants to have the right for manipulating the application being shared). The granted floor event message autonomously activates the fine-grained request action of the application which the requester wants to manipulate. Other nodes, on which other participants reside, are updated to reflect the current floor holder in the session. This mechanism shows a mapping instance of a universal social protocol (request-response) from an offline session to an online session with a set of user interfaces between a participant in a session and a collaboration environment as shown in Fig. 6. In our collaborative applications, whiteboard application uses this mechanism. Note that audio/video applications can use this request-response interaction scheme with the application specific locking mechanism as we integrate the scheme into the applications as a next phase in future work.

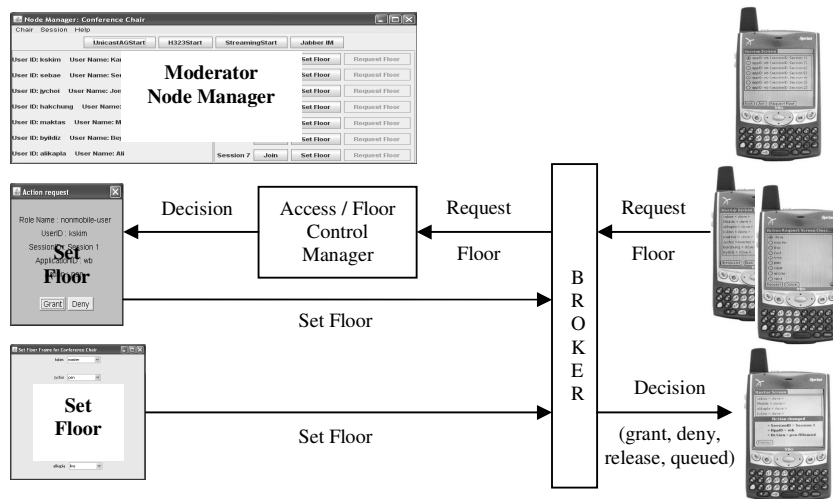


Fig. 6. Request-Response Interaction Scheme between a Moderator and a Floor Requester with Human-Computer Interaction

Chess game application uses different floor control mechanism able to be behaved without the mediation of the moderator like turn-taking mechanism. Thus, if one player in the chess game releases a floor or the prefixed playing time of a player is expired, then the floor is autonomously given to another player. In the two player game, the floor holder (one player in the game) directly passes the floor to another player through a broker. Other participants are regarded as an observer role which cannot have a floor to play but share playing-view in the chess game. Fig. 7 depicts the instance of the two-player turn-taking mechanism for the collaborative chess game application used in our collaboration.

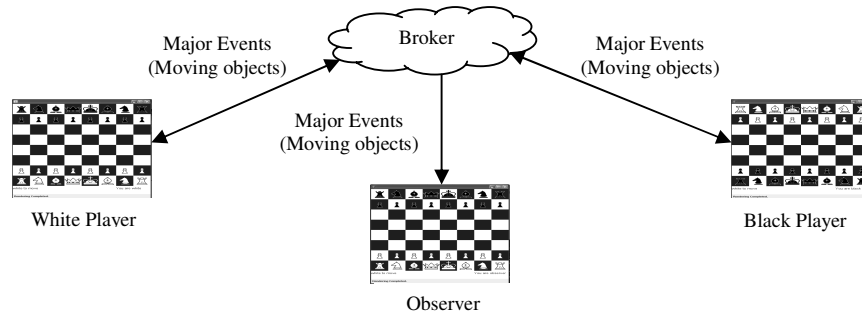


Fig. 7. Two-player Turn-taking Mechanism for Chess Game Application

5.3 Functionality of XGSP-Floor Control Tool

This section describes the functionality of a floor control tool (XGSP-Floor tool interfaces) that provides a user interface for control of floors to a moderator and participants in a session with desktop and cell phone devices. A moderator can give decisions (grant, deny, release (or revoke), and queued) for a floor to participants and the participants can request a floor to manipulate the application being shared in a session via the XGSP-Floor tool.

Fig. 8 shows a node manager of a moderator on desktop. Fig. 9 shows a node manager of participants (normal users) on desktop. The two node managers are almost similar except that the moderator node manager has a button able to control a floor and the normal user node manager has a button able to request a floor. Therefore, participants (not moderator) are not able to control the floor to give the right for manipulating the application being shared to other participants. The left display panel in the node managers shows a list of participants joined in the conference. The right display panel shows a list of sessions available in the conference. Each entry in a list of sessions has a session ID and three buttons (Join, Set Floor, and Request Floor). Participants in a conference can join a session by clicking on the “Join” button. A moderator can control floors in the window frame invoked by clicking on the “Set Floor” button in Fig. 8. The pop-up window frame is shown in the left figure of Fig. 10. Participants can request floors in the window frame invoked by clicking on the “Request Floor” button in Fig. 9. The pop-up window frame for the request floor is shown in the right figure of Fig. 10. A moderator can control floors of all the participants joined in a session via the window frame shown in the left figure of Fig. 10 while participants can request only their own floors via the window frame shown in the right figure of Fig. 10 but see current floor states of other participants.



Fig. 8. Node Manager for a Moderator on Desktop



Fig. 9. Node Manager for Normal Users on Desktop



Fig. 10. Set Floor Frame for a Moderator vs. Request Floor Frame for a Normal User



Fig. 11. Node Manager for Normal Users on Cell Phone

Fig. 11 shows a node manager of normal users (nomadic users) on cell phone. The functionality of the node manager on cell phone is similar to that of the node manager on desktop except that the node manager on cell phone uses two different screens for the presence membership of participants and a list of sessions existing in a conference. The left figure in Fig. 11 shows a list of participants joined in the conference. The right figure shows a list of sessions available in the conference. Note that the cell phone uses the term screen instead of the term window or frame used in desktop. The application model for pervasive computing [1] requires creating a task-based model and a navigation model for program structure at design time. This means the task-based structure needs to generate device specific “presentation units” – screen and to specify the flow of the presentation units. Therefore, an application has to be depicted into tasks, subtasks of the tasks, and subtasks of the subtasks, and so on. On the modest-size window like desktop, the tasks (displays of participants’ presence panel, session panel, floor set window frame, floor request window frame, and so on) in node manager can be presented on the same screen, whereas on the small-size screen like cell phone, the displays of the tasks have to be presented on separate screens including easy-to-use interfaces and a set of well-defined navigation to subtasks from the tasks. Fig. 12 shows the request screen for a floor. The screen is displayed from selecting the “Request Floor” button shown in the right figure of Fig. 11. Fig. 13 shows pop-up window frames (floor request, floor grant, floor deny, floor conflict, and floor queued notifications respectively) occurred as the request-response interaction mechanism between a moderator node and a requester node on desktop is used. Fig. 14 shows the screens (floor grant, floor deny, and floor queued notifications respectively) occurred as the request-response interaction mechanism between a moderator node

and a requester node on cell phone is used. Then the screen in cell phone is often called an alert screen that shows a message to the user for a certain period of time.



Fig. 12. Request Floor Screen on Cell Phone



Fig. 13. Pop-up Window Frames (for Floor Request, Floor Grant, Floor Deny, Floor Conflict, and Floor Queued Notifications respectively) on Desktop



Fig. 14. Screens (for Floor Grant, Floor Deny, and Floor Queued Notifications respectively) on Cell Phone

In this section we showed the current implementation of an XGSP-Floor tool. It shows a simple interface between participants and collaboration environment for floor request, response, release (or revoke), and queued interaction with the synchronous collaborative application – shared whiteboard used in our collaboration. We need to further implement the XGSP-Floor tool with more detailed functionalities for synchronous collaborative media applications such as audio and video applications in future work.

5.4 A Major Event Conflict Detection Function of XGSP-Floor Mechanism

This section describes a major event conflict detection function that determines whether an action in a floor request conflicts with the action of current floor holder. When a floor is requested at the application being shared among participants, it consults with the current floor state information table in the access and floor control manager shown in Fig. 5 to avoid the collision with current

floor holder. If a request action exists in the current floor state information table, then the request is queued. Otherwise, the request is granted. The floor state information is updated to reflect an action in a floor request whenever the request is granted. Participants maintained in the floor state information table have to assume at least one action but cannot assume both actions at the same time even though the participants can assume different actions at the different time in our current floor mechanism. Participants in passive state may assume the action “slave” in our collaboration. The action “slave” means participants are in state joined in a session and in view-sharing state of the application being shared with other participants for WYSIWIS (What You See Is What I See) [38] which is an inclusion of collaboration. Also, the action can be used as a major event for releasing a floor which a participant is currently holding. The strict conflict avoidance [11] like our floor control mechanism allows all participants to have the same views and data at all times. Pessimistic (or non-optimistic) floor control follows the strict conflict avoidance strategy whereas optimistic floor control strategy allows conflicts and resolves them [8].

5.5 Locking of XGSP-Floor Mechanism

Locking [19] is a method of gaining privileged access to shared resource for some amounts of duration. In this section we show non-optimistic locking mechanism [19] used in our synchronous collaboration from two different viewpoints, moderator’s point of view (system’s point of view) and participant’s point of view (application’s point of view), where the non-optimistic locking mechanism means a request node (or a requesting participant) has to wait until the floor request gets granted from the moderator. This non-optimistic locking mechanism ensures that all the participants always have consistent views and data. From moderator’s point of view or system’s point of view, the floor request queue in a moderator node is locked until the moderator node (or moderator) makes a decision on a floor request and dispatches the decision to the request node. During the lock, the floor state information table is updated. After the lock of the floor request queue is released, the next request in the queue is serviced if the queue is not empty. This locking mechanism guarantees the mitigation of race conditions of floor requests to shared application and thus enforces mutual exclusion in the shared application. Therefore, participants can access a shared application with a granted fine-grained action one participant at a time.

From participant’s point of view or application’s point of view, we used an application specific locking mechanism. According to shared applications, different fine-grained locks are used to allow more concurrent activity among participants and to follow the principle of least privilege [29], where the fine-grained lock means the locking of the major event described in section 5.1.2. Also, a coarse-grained lock can be used to allow a participant to make more activities at a time. In our whiteboard application example, as a fine-grained request action (major event) is granted from a moderator and the granted message arrives at the requester node, the lock for the use of the requesting action is released as depicted in Fig. 15. The coarse-grained action “master” in the application can be used to allow a participant to assume many different fine-grained actions at a time. This locking mechanism guarantees that the consistent state at application level is maintained among participants. In our chess game application example, if an action (a major event moving a object) of a player is a legal move validated by the chess game rule as the user input event (releasing the object – mouse release) of the player is occurred, then the user input event (mouse click) of the player is locked and the lock (mouse click) of another player is released in turn by passing a logical token as depicted in Fig. 16.

In our first implementation, we used a mechanism: it does not use a reply message from a requester node for a lock release of the floor request queue in a moderator node (no acknowledgement) and the action, which the requester is currently holding, is not blocked (non-blocking). This means the requester can manipulate shared application with a holding action until a grant message for new floor arrives at the requester node. Then, we identified the mechanism results in inconsistency. The inconsistency comes from: before the floor of current floor holder for shared application is revoked, the floor can be reassigned to another participant.

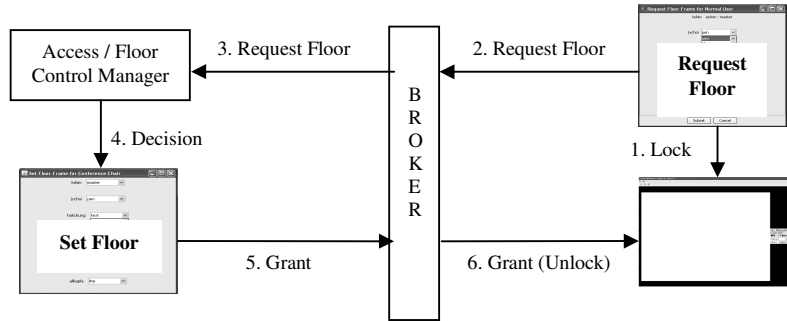


Fig. 15. Locking Mechanism of Shared Whiteboard

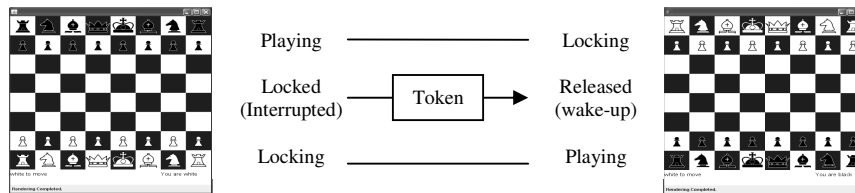


Fig. 16. Locking Mechanism by Logical Token-passing in Chess Game

In our second implementation, we used an acknowledgement (reply message) from a requester node after a request-response interaction for a floor between a request node and a moderator node. The implementation with the acknowledgement from the requester node ensures that the floor of current floor holder for shared application is revoked before the floor is reassigned to another participant and thus previous floor holder no longer holds the floor of newly reassigned current floor holder. Also, a floor requester is assigned a floor after current floor state information of the requester is updated. The acknowledgement enforces mutual exclusion in shared application and thus ensures consistency. But it resulted in response overhead, especially with cell phone device involved in collaboration since the wireless network transit time is in the range of seconds [24]. Another encountered problem was deadlock occurred from the loss of the acknowledgement (lock release of a moderator node), especially with cell phone disconnected. We could resolve the problem by a moderator directed-floor-assignment or communication among participants through text chat using no floor control mechanism. Then the lock for mitigating floor requests in a moderator node is released and the next request can be served. Our current floor control mechanism implements no acknowledgement and blocking mechanism where blocking means the action, which a floor requester is currently holding when she makes a request for a floor, is blocked if she holds a floor. But, the mechanism uses the acknowledgement (reply message) from revoked node to prevent the floor of current floor holder from assigned to another participant before the floor of the floor holder is revoked.

6. FORMAL VERIFICATION OF XGSP-FLOOR CONTROL MECHANISM BY COLORED PETRI NET

This section shows modeling of XGSP-Floor control mechanism. We also show formal verification to prove the correctness of the modeled XGSP-Floor control mechanism in terms of mutual exclusion, dead lock, and starvation. The key part for the modeling and formal verification of the modeled control mechanism is to show consistent shared state at application level to collaborating users by mitigating race conditions for shared resources and thus to attain mutual exclusion among resources. For the abstract modeling representation of the control mechanism, we used Colored Petri Nets (CP-nets or CPNs) with time [26]. The CP-nets provides a formal simulation tool [6] to model a system. In our modeling, the simulation involves programming the model of the control mechanism. Data structures in the simulation represent the major components of the control mechanism. The CP-nets also provides analysis functions using state spaces (also called occurrence graphs) to prove the correctness of a system based on mathematical methods. As the simulation executes, the simulation tool updates the simulation

state to reflect the activities of the modeled control mechanism and a set of statistical data are gathered and used to prove the correctness of the modeled control mechanism.

6.1 Modeling of XGSP-Floor Control Mechanism

In this section we present decision procedures behaved in a moderator node (which is a decision node to control accesses for resources in our collaboration system) to determine grant or deny of collaborating users' requests to access resources in our XGSP-Floor control mechanism. The decision procedures of the moderator node (or by a moderator) are modeled in terms of the following five different types of stages. The broad view of the modeling for the XGSP-Floor is shown in Fig. 17.

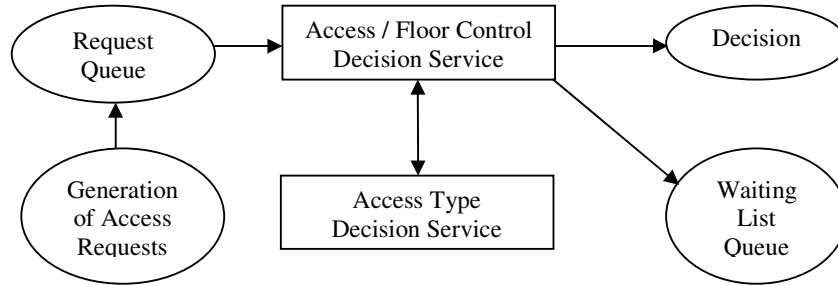


Fig. 17. Broad View of the Modeling for XGSP-Floor

First, the modeling randomly generates access requests to resources by the simulation on behalf of collaborating users in a collaboration session and has a single request queue for storing access requests from the simulated users. The queue is implemented in FIFO (First-In, First-Out) order for mitigating race conditions of access requests to resources. The first request in the queue is sent to the access type decision service located in external process module for parsing the requests written in XML and returning a type value into the modeling. Then, the first request is removed from the queue. During the activity, new access requests are generated and stored in the request queue waiting for next service.

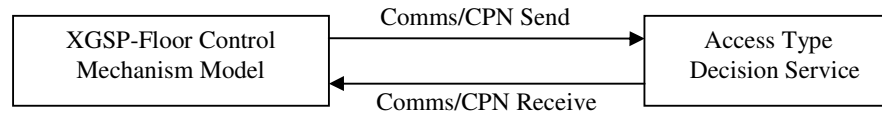


Fig. 18. Comms/CPN for Communication between Control Mechanism Model and Access Type Decision Service which is located outside the modeling as external process

Second stage is a communication activity for accessing the access type decision service, mentioned above, which is located in external process module outside the XGSP-Floor mechanism model. For the communication activity, CP-nets provides Comms/CPN [14, 15] which is a library for communication service between CP-nets models and external processes. The Comms/CPN library allows CP-nets models to interact with the external environments via TCP/IP. Using the library, the control mechanism model connects to the access type decision service which is a module written in Java 1.5 and which is practically used in XGSP-Floor control mechanism integrated into our collaboration framework. The service parses XML requests sent from the control mechanism model and returns a type value among Invalid, Implicit, Exclusive, Shared, or Released into the model as practically does the access type decision service module in our collaboration framework. Fig. 18 shows the communication between the control mechanism model and the access type decision service. In the figure, the Comms/CPN send function is used to send access requests written in XML to the external type decision service and the Comms/CPN receive function is used to receive the response from the external service.

Third, decision activities in the modeling are behaved with a type value returned from the access type decision service. The decision activities are also branched into the same access type activities with a returned value as mentioned in the second stage. Each decision activity simulates decision behaviors (grant, deny, released or queued) of a moderator in collaboration by randomly generating access decisions for requests in the modeling.

Next stage is to update a state information table of a requesting user with simulated decision behaviors. Some requests are denied without need for updating current action state information of the requesting users, some requests are granted with need for updating the current action state of users, or others are stored in a queue (waiting list queue for access to shared resources) different from the request queue storing access requests to resources. Finally, all the requests stored in a queue for the use of resources are serviced in prefixed amount of time to avoid starvation. The access requests to shared resources are stored in a single queue which is implemented in FIFO (First-In, First-Out) order which has a fair characteristic. The first request in the queue is serviced when a floor holding a shared resource is released or after an appropriate amount of time. Then, the request is removed from the queue and the current state information table is updated with the removed request action.

Also, we define a set of predicate rules used as determinants in decision procedures of a moderator node in terms of the following two types of predicate statements:

- 1 Determination of types classified to access resources.
This predicate statement is similar to the statement described in section 5.2.1, except that 4-tuple (userID, roleName, applicationID, action) for requesting the use of a resource is used whereas 2-tuple (userID, action) is used in our practical mechanism., where roleName means the name of roles assigned to users in our collaboration system, applicationID means an application identifier existing in application registries of our collaboration system.
- 2 Determination of whether an action in a request exists in state information table, in other words, a request action conflicts with the action of current floor holder.
This predicate statement is also similar to the statement described in section 5.2.1.

6.2 Informal Introduction of XGSP-Floor Modeled by Colored Petri Nets

In this section, we informally show how CP-nets with time is able to be used to model the XGSP-Floor control mechanism. The basic idea for modeling of the XGSP-Floor using CP-nets with time is to describe a method for mitigating race conditions of access requests to resources, and thus ensuring shared state consistency at application level by attaining mutual exclusion among resources. The CP-nets model of the XGSP-Floor is depicted in Fig. 19. The XGSP-Floor control mechanism is modeled by means of places, transitions, and arc expressions between the places and the transitions. We present an informal introduction of the modeled mechanism in terms of places, transitions, and arc expressions. Also we show the correctness of the modeled XGSP-Floor control mechanism with the informal definition in terms of mutual exclusion and starvation.

Each place in the model has a color set (interchangeably often referred to as a data type). The color set determines a kind of data type which places can have. A value of a color set is called a token color in CP-nets as an element of a data type in high-level programming language is called a value of the type. From Fig. 19, it can be seen that the places Simulation-Start and Request-Nodes have a color set COUNT and the places Time and Waiting-Time have a color set INT. The place Request-Queue has a color set NewReqs, the place State-Information-Table has a color set OldReqs, and the place Waiting-List-Queue has a color set SharedReqs. Also, the places Busy, Invalid, Implicit, Shared, Exclusive, and Released have a color set LockxNewReq. The places Next-Request, L, and GiveFloor have a color set Lock. The place U has a color set LockxGrantDeny. The place TimeOver has a color set BOOL. Other places have a color set LockxGrantDenyxNewReq. For example, the place Z is used to send a decision on an access request to a request node. Then the place Z in our modeling can have values which are composed of a value from Lock color set, a value from GrantDeny color set, and a value from NewReq color set where the color set Lock is a data type used to unlock decision procedures, the color set GrantDeny is a data type which can have a value from {grant, deny, queued, released, give}, and the color set NewReq is a data type of new request which the place can have. The informal descriptive definitions of the color sets in the modeling of the XGSP-Floor control mechanism are as follows.

COUNT = {0@time, 1@time, 2@time...} where @time means some model time.

INT = {0, 1, 2, 3 ...}

smallINT = {0, 1, 2, 3, 4}

BOOL = {true, false}

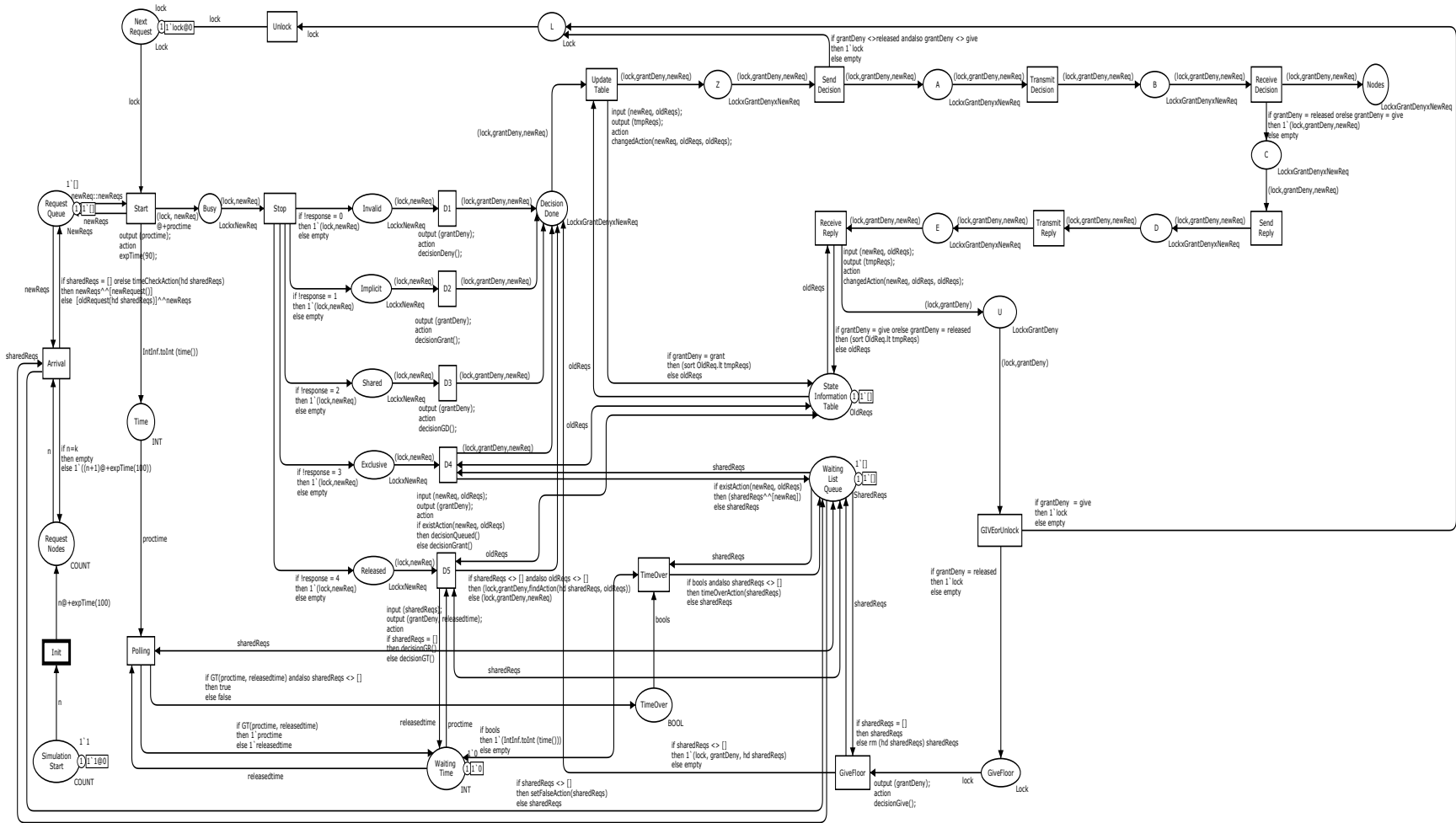
UserID = {A, B... J}

RoleName = {nonmobile_user, mobile_user}
AppID = {wb}
Action = {master, slave, line, rect, oval, pen, eraser, move, load, clear}
NewReq = {(i, j, k, l, m) | i ∈ UserID and j ∈ RoleName and k ∈ AppID and l ∈ Action and m ∈ BOOL}
OldReq and **SharedReq** = same as NewReq
NewReqs = [{(i, j, k, l, m) | i ∈ UserID and j ∈ RoleName and k ∈ AppID and l ∈ Action and m ∈ BOOL}, {.....}, {.....}, ...]
OldReqs and **SharedReqs** = same as NewReqs
Lock = {lock@time where lock is a variable used to lock decision procedure and @time means some model time}
LockxNewReq = {(i, j) | i ∈ Lock and j ∈ NewReq}
GrantDeny = {grant, deny, queued, released, give}
GrantDeny2 = {grant, deny}
LockxGrantDenyxNewReq = {(i, j, k) | i ∈ Lock and j ∈ GrantDeny and k ∈ NewReq}
LockxGrantDeny = {(i, j) | i ∈ Lock and j ∈ GrantDeny}

A state (often referred to as marking in CP-nets) of a place represents current state of token colors of color sets which the place has. For example, the current and initial marking of the place Simulation-Start in Fig. 19 is COUNT. This means the place can have token colors from arbitrary natural numbers beginning with 1. The current value of the place is used as initial value (or token color) to count the number of requests for accessing resources from simulated users. The initial markings of places Request-Queue, Next-Request, Waiting-Time, Waiting-List-Queue, and State-Information-Table from Fig. 19 represent initial token value of the color sets which the places have. The place Next-Request has an initial token color lock with timed type and the place Waiting-Time has an initial token color 0 (zero) with integer type. The initial markings of the places Waiting-List-Queue and State-Information-Table have a token color with empty list which means the number of elements in the list is zero respectively. All other places initially have empty which means the places have no token colors.

Token removed from incoming places are transferred to outgoing places by evaluating arc expressions occurred by the transition connected to the places. For example, in Fig. 19, the transition Init has one incoming arc and one outgoing arc. The arc expression of incoming arc into the transition is n where n is a variable of a color set COUNT in the place Simulation-Start. The value of the CP-nets variable n is 1 since the token value which the place Simulation-Start has is 1. And the arc expression of outgoing arc from the transition Init is a variable n and function expTime (100) where n is also a CP-nets variable and has the same value 1 as the value of the color set COUNT transferred into the transition, and expTime (100) is a function to exponentially calculate some delay time in the interarrival requests and is used to simulate the requests as if collaborating users practically behave to request accesses to resources. The interarrival requests' times are exponentially distributed with a mean of 100 time units between two successive requests issued by the simulation tool. So the delay time has no any meaning and just is used to randomly generate independent requests. The enabled transitions are usually occurred by evaluating outgoing arcs as a previous instance. Then, a binding element, which is composed of an enabled transition and a binding of outgoing arcs, has to be considered to evaluate the outgoing arcs connected from the transition. Also, as another instance in Fig. 19, each of the transitions D1, D2, and D3 has one incoming arc and one outgoing arc. The arc expression of the incoming arc is (lock, newReq) where lock is a variable of color set Lock and newReq is a variable of color set NewReq. And the arc expression of outgoing arc is (lock, grantDeny, newReq) where lock is a variable of color set Lock, grantDeny is a variable of color set GrantDeny and newReq is a variable of color set NewReq. Assume that a global variable - response has now a value 2. Then the place Shared has a token value (lock, newReq) and the transition D3 is enabled. With a decision value returned from the function decisionGD(), the transition D3 binds the expression (lock, grantDeny, newReq) of the outgoing arc to (lock, grant or deny, newReq). Therefore, the place has a token value which is composed of a value of lock variable, a value of grantDeny variable, and a value of newReq variable. Thus, in such sequences of occurrences, the occurrence of a transition simulates decision procedures of the control mechanism. All other transitions in the model of the control mechanism are enabled and occurred in such a similar way.

Fig. 19. Control Mechanisms Modeled by CP-nets



To show the correctness of the modeled mechanism in terms of mutual exclusion, we consider the places Request-Queue and Next-Request, and the transition Start. A request token from the place Request-Queue and a lock token from the place Next-Request enable the transition Start. When the transition is occurred, the two tokens are added to the place Busy by evaluating the arc expression between the transition Start and the place Busy, and the tokens (a request token and a lock token) are removed from the places Request-Queue and Next-Request. The transition Start then will not be enabled until the place Next-Request has a new token value lock, where the token value lock is generated after the state information table is updated and a decision on a request is sent to a request node. Therefore, the following requests are not able to enter the decision procedures which are regarded as a critical section in the modeling until the place Next-Request has a new token. This means at most one request is processed in the decision procedural stage and thus indicates the modeled control mechanism is ensuring mutual exclusion for the decision procedural stage to avoid race condition among requests issued by the simulation tool on behalf of users.

Next, we consider the transitions Start, Polling, and TimeOver to show that there is no starvation among requests issued in the modeling. When the transition Start is occurred, the transitions Polling and TimeOver check the time duration of the requests waiting for floors in the place Waiting-List-Queue. If the waiting time duration of a request is over the prefixed amount of time, then the request is serviced. Thus, the requests waiting for floors in the place Waiting-List-Queue will never be starved.

6.3 Formal Definitions and Notations of XGSP-Floor Control Mechanism Modeled by Colored Petri Nets

In this section we present a formal definition of static structure and dynamic behavioral properties that CP-nets has, and the representation of the static properties and the example representation of the dynamic behavioral properties for the CP-nets model of our XGSP-Floor control mechanism.

6.3.1 Static Structure Properties of CP-nets and Representation of XGSP-Floor by the Properties

The static structure is basically composed of building blocks (places, transitions, and arcs), the connection points through which data flow into and out of the building blocks, and the connection paths along which data flow between the building blocks.

The static properties [26] of the CP-nets are represented as a 9-tuple $(\Sigma, P, T, A, N, C, G, E, I)$ where

1. Σ is a finite set of types (also, called color sets).
2. P is a finite set of places.
3. T is a finite set of transitions.
4. A is a finite set of arcs such that $P \cap T = P \cap A = T \cap A = \emptyset$.
5. N is a node function: $A \rightarrow (P \times T) \cup (T \times P)$.
6. C is a color function: $P \rightarrow \Sigma$.
7. G is a guard function such that $\forall t \in T$ $[\text{Type}(G(t)) = \text{Bool} \wedge \text{Type}(\text{Var}(G(t))) \subseteq \Sigma]$ where $\text{Type}(G(t))$ denotes the type of $G(t)$, Bool denotes $\{\text{true}, \text{false}\}$ and $\text{Var}(G(t))$ denotes the set of variables in $G(t)$.
8. E is an arc expression function such that $\forall a \in A$ $[\text{Type}(E(a)) = C(p(a))_{MS} \wedge \text{Type}(\text{Var}(E(a))) \subseteq \Sigma]$ where $p(a)$ is the place of $N(a)$, $\text{Type}(E(a))$ denotes the type of $E(a)$, $C(p(a))_{MS}$ denotes the set of multisets over a set $C(p(a))$, and $\text{Var}(E(a))$ denotes the set of variables in $E(a)$.
9. I is an initialization function such that $\forall p \in P$ $[\text{Type}(I(p)) = C(p)_{MS}]$ where $\text{Type}(I(p))$ denotes the type of $I(p)$ and $C(p)_{MS}$ denotes the set of multisets over a set $C(p)$.

From the set of color sets expressed Σ in above 9-tuple, the XGSP-Floor control mechanism has the set of color sets as follows.

$$\Sigma = \{\text{COUNT}, \text{INT}, \text{UNIT}, \text{BOOL}, \text{smallINT}, \text{Lock}, \text{GrantDeny}, \text{GrantDeny2}, \text{UserID}, \text{RoleName}, \text{AppID}, \text{Action}, \text{NewReq}, \text{OldReq}, \text{SharedReq}, \text{NewReqs}, \text{OldReqs}, \text{SharedReqs}, \text{LockxNewReq}, \text{LockxGrantDenyxNewReq}, \text{LockxGrantDeny}\}$$

The elements P , T , and A in the 9-tuple are a set of places, transitions, and arcs respectively. The N means no arc may connect two places or two transitions. In the CP-nets model of XGSP-Floor

control mechanism, the color function C maps the places Simulation-Start and Request-Nodes into COUNT, the place Request-Queue into NewReqs, the places Time and Waiting-Time into INT, the place State-Information-Table into OldReqs, the place Waiting-List-Queue into SharedReqs, the places Busy, Invalid, Implicit, Shared, Exclusive, and Released into LockxNewReq, the places Next-Request, L, and GiveFloor into Lock, the place U into LockxGrantDeny, the place TimeOver into BOOL, and all other places into LockxGrantDenyxNewReq. Item7, the guard function is an expression which evaluate to Boolean (true or false). The arc expression function and initialization function are also an expression which evaluate to valid type value. The declarations for the CP-nets model of XGSP-Floor control mechanism are represented using the CP-nets ML (an extension of the functional programming language SML (Standard Meta Language)) [23] in Fig. 20.

Declaration of Variables		
val k = 1000;	val queuedTime = 100;	var bools : BOOL;
var n : COUNT;	var decisionNum : smallINT;	var proctime : INT;
var releasedtime : INT;	var newReq : NewReq;	var oldReq : OldReq;
var sharedReq : SharedReq;	var newReqs : NewReqs;	var oldReqs : OldReqs;
var tmpReqs : OldReqs;	var sharedReqs : SharedReqs;	
var grantDeny : GrantDeny;	var grantDeny2 : GrantDeny;	
globref connected = false;	globref response = 0 : smallINT;	

Declaration of Color sets		
colset BOOL = bool;	colset COUNT = int timed;	colset UNIT = unit timed;
colset INT = int;	colset smallINT = int with 0..4;	colset Lock = with lock timed;
colset UserID = with A B C D E F G H I J;		
colset RoleName = with nonmobile_user mobile_user;	colset AppID = with wb;	
colset Action = with master slave line rect oval pen eraser move load clear;		
colset NewReq = record userID : UserID * roleName : RoleName * appID : AppID * actions : Action * AT : BOOL;		
colset OldReq = record userID : UserID * roleName : RoleName * appID : AppID * actions : Action * AT : BOOL;		
colset SharedReq = record userID : UserID * roleName : RoleName * appID : AppID * actions : Action * AT : BOOL;		
colset NewReqs = list NewReq;		
colset OldReqs = list OldReq;		
colset SharedReqs = list SharedReq;		
colset LockxNewReq = product Lock * NewReq timed;		
colset GrantDeny = with grant deny queued released give;		
colset GrantDeny2 = subset GrantDeny with [grant , deny];		
colset LockxGrantDenyxNewReq = product Lock * GrantDeny * NewReq;		
colset LockxGrantDeny = product Lock * GrantDeny;		

Functions	
fun decisionGrant () = grant;	fun decisionDeny () = deny;
fun decisionQueued () = queued;	fun decisionGD () = GrantDeny2.ran();
fun decisionGive () = give;	fun intTime () = IntInf.toInt (time());
fun decisionGT () =(released, IntInf.toInt (time()));	
fun decisionGR () = (grant, IntInf.toInt (time()));	
fun GT(a:int, b:int) = ((a-b) > queuedTime);	
fun changedAction(newReq:NewReq, tmpReqs:OldReqs, oldReqs:OldReqs) = (if length oldReqs = 0 orelse length tmpReqs = 0 then oldReqs^^[newReq] else (if (#userID newReq) = (#userID (hd tmpReqs)) then (rm (hd tmpReqs) oldReqs^^[newReq] else changedAction(newReq, tl tmpReqs, oldReqs)))	
fun existAction(newReq:NewReq, oldReqs:OldReqs) = (if length oldReqs = 0 then false else (if (#actions newReq) = (#actions (hd oldReqs)) then true else existAction(newReq, tl oldReqs)))	
fun findAction(sharedReq:SharedReq, tmpReqs:OldReqs) = (if length tmpReqs = 0 then sharedReq else (if (#actions sharedReq) = (#actions (hd tmpReqs)) then {userID = #userID (hd tmpReqs), roleName = #roleName (hd tmpReqs),	

```

    appID = #appID (hd tmpReqs), actions = slave, AT = #AT (hd tmpReqs))
    else findAction(sharedReq, tl tmpReqs)))
fun timeCheckAction(sharedReq:SharedReq) =
  ( if (#AT sharedReq) then false else true)
fun timeOverAction(sharedReqs:SharedReqs) =
  ( [{userID = #userID (hd sharedReqs), roleName = #roleName (hd sharedReqs),
    appID = #appID (hd sharedReqs), actions = #actions (hd sharedReqs),
    AT = true}]^rm (hd sharedReqs) sharedReqs)
fun setFalseAction(sharedReqs:SharedReqs) =
  ( [{userID = #userID (hd sharedReqs), roleName = #roleName (hd sharedReqs),
    appID = #appID (hd sharedReqs), actions = #actions (hd sharedReqs),
    AT = false}]^rm (hd sharedReqs) sharedReqs)
fun expTime (mean: int) =
  let val realMean = Real.fromInt mean
      val rv = exponential((1.0/realMean))
  in floor (rv+0.5)
  end;
fun newRequest() =
  {userID = UserID.ran(), roleName = RoleName.ran(),
  appID = AppID.ran(), actions = Action.ran(), AT = false}
fun oldRequest(sharedReq:SharedReq) =
  {userID = #userID sharedReq, roleName = #roleName sharedReq,
  appID = #appID sharedReq, actions = slave, AT = #AT sharedReq}
fun get_from_externalProcess() =
  ConnManagementLayer.receive("Conn", integerDecode);
fun send_to_decisionService(roleName, appID, action) =
  ConnManagementLayer.send("Conn",
"<RoleName>"^roleName^"</RoleName><AppID>"^appID^"</AppID>
<Action>"^action^"</Action>", stringEncode);

```

Fig. 20. Declarations for the CP-nets model of Control Mechanism

6.3.2 Dynamic Behavioral Properties of CP-nets and Representation of XGSP-Floor Control Mechanism by the Properties

The dynamic behavior of CP-nets is a data transformation between the occurring transition and the occurred transition with a time delay of some small magnitude. In this section, we present the dynamic behavioral properties of CP-nets [26] about binding, marking, enabling, and occurrence, and the example representation of them in the modeled XGSP-Floor control mechanism.

- Binding – this means to bind correct token values to the variable of the token type. For example, in our modeling, the transition Start may have the binding element such as <Start, (lock, {userID = kakim, roleName = mobile-user, appID = wb, action = pen, AT = false})@777> which is composed of a transition and a binding.
- Marking – a set of multisets over the set of tokens positioned on the individual places. For example, the places Request-Queue, Waiting-List-Queue, and State-Information-Table have an empty list token as an initial marking respectively.
- Enabling – when tokens from all the input places of a transition are evaluated by the arc expressions between the input places and the transition and before the tokens are added to the output places of the transition, the transition is called enabled with a set of binding elements.
- Occurrence – after the transition is enabled and by removing token from the input places and adding the tokens to the output places of the transition, the transition is called occurred and then the occurrence sequence is composed of a sequence of reachable markings and occurring steps. For example, when the transition Start occurs, one specified token will be removed from the input places Request-Queue and Next-Request. At the same time, three tokens will be added to the output places. The place Request-Queue will get a token with a list type as a token color set, the place Busy will get a token with a value (lock, {userID = kakim, roleName = mobile-user, appID = wb, action = pen, AT = false})@777, and the place Time will get a token with the value of current modeling time.

6.4 Verification for Correctness of XGSP-Floor Control Mechanism based on State Space Analysis

In this section, we verify the correctness of the XGSP-Floor control mechanism modeled by the CP-nets from the previous sections with a means of simulations and state spaces [26]. The CP-nets provides a simulation tool [6] that simulates a system by nondeterministic distributing color tokens into a model, and a state space generation tool [6] that generates a report for a sequence of occurrence states. To construct state spaces means to generate all the possible occurrence graphs that are composed of nodes and arcs. Nodes in state spaces are generated for each reachable markings, and arcs in the state spaces are generated for each occurring binding elements. The report generated from the state space generation tool is used for verifying the correctness of a model. In the next five subsections, we analyze the simulation behaviors and the occurrence state information generated from the state space generation tool.

6.4.1 Statistical Information of State Spaces and SCC Graph

The state spaces report has the following five parts. The first part, statistical information report of state spaces and SCC (strongly connected components) graph [26], is shown in Table 1. The statistical report contains information about the size of nodes and arcs, and time and calculation status took for generating state spaces and SCC graphs. In the case of 4133 which the number of requests is, the state space has 52643 nodes and 79809 arcs in partially calculated graphs, and this took 300 seconds for generating the state space which is composed of the nodes and the arcs. Also, the report shows information of SCC graph that is identical to the information of the state space except for time taken for generating the components. The strongly connected components are a maximal subgraph to find a path from any one node to any other node. In the report, the number of strongly connected components in the modeled control mechanism is equal to the number of state space nodes. This implies that the modeled control mechanism has strongly connected components with just one node. Therefore, the modeled mechanism has no infinite occurrence sequences. This means the simulation of the modeled mechanism terminates after processing some number of requests if we put the stop criteria such as limiting the number of requests into the model. In other words it shows the modeled mechanism is working as we expect to achieve termination normally in forcibly limiting the number of requests. This report shows statistical information about sizes of state spaces and strongly connected components of the state spaces generated from the tools of CP-nets.

Statistics		
# of Requests = 4133		
	State Space	SCC Graph
Nodes	52643	52643
Arcs	79809	79809
Seconds	300	16
Status	Partial	

Table 1. Statistical Information of State Space and Scc Graph

6.4.2 Boundedness Properties and Mutual Exclusion of Modeled Control Mechanism

The second part shows boundedness properties of the state spaces report in Table 2. The properties express the upper integer bounds which is the maximal number of tokens and the lower integer bounds which is the minimal number of tokens that the places in a modeling may have. In the integer bounded information of the modeled mechanism, the places Busy, Decision-Done, Exclusive, Implicit, Invalid, Next-Request, Released, Shared, L, and Z have one token in upper bound and zero token in lower bound. These are places in decision procedural stage of a moderator node that is a critical section for mutual exclusion. Note that the upper integer bound of the places Busy and Next-Request is 1. This implies if 1 is upper integer bound for the places, then at most one request is processed in the decision procedural stage (critical section) at any time. As a contradiction, if the upper integer bound of the place Busy is more than two, then the upper bound of the place Next-Request has to be at least more than two at any time in any reachable markings. But we see the upper bound of the place Next-Request is 1 and thus the place Busy is not able to contain more than two token at any time in each reachable marking. This indicates

the modeled XGSP-Floor control mechanism is ensuring mutual exclusion for the decision procedural stage to avoid race condition among requests.

Also, we need to show all the requests that wish to enter a critical section have to be serviced and only one request must enter the critical section. In other words, all the requests have to be serviced and thus not starved for getting the service. To show this property, we consider the transitions Arrival and Start, and the integer bounds of the places Request-Queue and Next-Request in Fig. 19. The occurring transition Arrival puts new requests into the place Request-Queue and then the first request in the queue enters the decision procedural stage (critical section) through the occurrence of the transition Start. As shown in Table 5, the place Request-Queue has exactly one token at any time during the execution of the modeling, and the place Next-Request has one token as upper integer bound. This means the transition Start will be enabled at any time during the execution of the modeling, and hence one request will enter the decision procedural stage. Thus, the requests will be serviced one by one as we expect. This report also tells us that the places State-Information-Table, and Waiting-List-Queue as well as the place Request-Queue in each reachable marking have exactly one token in upper and lower integer bounds. This means the places have one list token each used as a queue and the modeled mechanism in Fig. 19 is also working as expected.

Boundedness Properties		
Best Integer Bounds	Upper	Lower
A	2	0
B	3	0
Busy, C, D, E, L, U, Z	1	0
Decision_Done	1	0
Exclusive	1	0
GiveFloor	1	0
Implicit, Invalid	1	0
Next_Request	1	0
Nodes	14	0
Released, Shared	1	0
Request_Nodes	1	0
Request_Queue	1	1
Simulation_Start	1	0
State_Information_Table	1	1
Time	3	0
TimeOver	3	0
Waiting_List_Queue	1	1
Waiting_Time	1	1

Table 2. Boundedness Properties

6.4.3 Home Properties

The third part of the report provides information about home properties. The home markings in the home properties mean markings which can always be reached from all reachable markings [26]. The property in Table 3 shows that the initial marking of the modeled control mechanism in Fig. 19 is not a home marking because the initial marking is a marking for starting just the modeled mechanism by substituting a integer value one for the transition Init as a binding element of the transition, and hence any subsequent markings never return to the initial marking. Table 5 shows the fairness property of the state spaces report. It also provides information about how often different binding elements occur in each markings of the modeled mechanism. The property shows that the modeled mechanism has no infinite occurrence sequences. In other words, the modeled mechanism has no infinitely many different binding elements. These properties imply any subsequent markings are not able to reach the initial marking and the initial marking has no many different binding elements in the modeling. Therefore, these properties show that the modeled mechanism in Fig. 19 is working as expected.

Home Properties	
Home Markings	Initial Marking is not a home marking

Table 3. Home Properties

6.4.4 Liveness Properties of Modeled XGSP-Floor Control Mechanism

The fourth part of the report provides information about liveness properties. The dead transition instances in the properties of the report mean some transition instances which are never enabled in all reachable markings. Also, the live transition instances in the properties mean transition instances which can always be enabled at least once more in all reachable markings. The report in Table 4 shows the modeled control mechanism has no dead transition instances, and no live transition instances. This implies that each transition in the modeled mechanism is enabled in at least one marking among all reachable markings. This also tells us that there are no transition instances which can always be enabled at least once more in all reachable markings of the modeled mechanism. Thus, the report in Table 4 shows the modeled mechanism is correctly working as expected, and from the timed CP-nets and the properties of previous sections, new requests in exponentially distributed arbitrary interval are generated and thus no dead lock situation in which all the requests may be blocked is occurred.

Liveness Properties	
Dead Transition Instances	None
Live Transition Instances	None

Table 4. Liveness Properties of Modeled Control Mechanisms

6.4.5 Fairness and Starvation Properties of Modeled Control Mechanism

The fifth part of the report provides information about fairness properties. The information tells us how often the different binding elements of each transition in a modeling can occur [26]. The report in Table 5 shows there are no infinite occurrence sequences in the modeled mechanism. But the modeled mechanism may have infinite occurrence sequences in the transition Arrival of the modeling that has some binding elements (Arrival, NewReqs) which are repeated indefinitely where NewReqs is a type of color set with list type. The CP-nets does not consider such binding elements as making sense. Thus, the report shows the modeled mechanism has no infinite occurrence sequences. The trivial infinite occurrence sequences occurred in the transition Arrival also show if the transition ceases to occur, the simulation of the modeled mechanism must terminate. The transition generates new requests until some prefixed number of requests. Therefore, the properties also show the modeled mechanism is correctly working satisfying stop criteria to terminate the simulation as expected because there are no enabled transitions in the modeling after the transition Arrival generates some prefixed number of requests, i.e. there are no more enabling and occurrence in the transition Arrival. In addition to the expectation, the modeled mechanism is fair. Any requests may never be starved since there are no infinite occurrence sequences which may starve the requests forever. Also, the requests waiting for floors in the place Waiting-List-Queue in Fig. 19 will never be starved. Thus, it shows there is no starvation in the modeled control mechanism.

Fairness Properties
No infinite occurrence sequences

Table 5. Fairness Properties

7. Summary and future work

In this paper we have attempted to provide the virtual workspace with floor control capability and collaborative applications for synchronous and ubiquitous collaboration. This attempt has been driven by building integrated collaboration system including cell phone devices – Palm OS 5.2.1H Powered Treo600 [40]. As ubiquitous collaboration and access becomes more prevalent in the future, it will become more important to provide a virtual workspace in which geographically dispersed users can work together.

In this paper, we also presented a policy and a mechanism implementing it – XGSP-Floor (XGSP Floor control) for coordinating concurrent activities to synchronous collaborative applications and maintaining shared state consistency at application level. The XGSP-Floor mechanism uses moderator-mediated interaction with a major event conflict detection function and non-optimistic locking mechanism. The XGSP-Floor integrated into our collaboration framework provides significant flexibility, ranging from free-for-all (no floor) to application specific floor control

mechanism for avoiding uncoordinated activities to shared collaboration applications. A moderator (moderator node) is responsible for maintaining the consistent state of applications in our collaboration system. Even though our underlying floor control scheme is a moderator-mediated interaction mechanism, a floor can automatically be assigned to a floor requester without the mediation of the moderator according to the policy. We showed with example applications – shared whiteboard and collaborative chess game that social protocols used in a face-to-face offline session can be mapped to mechanisms able to be used in an online session with user interfaces between participants and CSCW environment. The XGSP-Floor control tool provides human-computer interaction for control of floor for roaming participants with cell phone as well as desktop participants in collaboration.

During our experiments with the collaboration framework, one of problems encountered was a failure like network disconnection of a moderator or chairperson node. If a moderator or chairperson node fails or is disconnected, and is not able to recover from the failure for some amount of time, one of participants in collaboration capable of having the role capability of the moderator or chairperson has to be elected. We tested it with an event driven message mechanism. But, when the network connection of a moderator or chairperson node was lost, it did not work since the event messages could not be disseminated in disconnected network. One approach to overcome the problem by exploring different fault-tolerant role delegation mechanism (for example, polling mechanism by heart-beat message between a moderator node and a conference manager) with role hierarchy policy will be considered in future work. We also left in future work support of the role hierarchy policy with the fault-tolerant role delegation mechanism issue.

CGL has an interactive two-player collaboration chess game for desktop devices. We did not implement the chess game on cell phone. In future work, we will consider the design and implementation of the chess game with shared event model on Wi-Fi (wireless fidelity) [42] phone, to reduce latency and to improve computing capability. Also we will consider the extension of our work (collaboration framework and other collaborative applications as well as the collaborative chess game) to new generation of cell phone such as iPhone 3G [22] which is multimedia and Internet-enabled mobile phone.

ACKNOWLEDGMENTS

Many thanks to my former colleagues at Community Grids Lab in Pervasive Technology Labs in Indiana University who developed the earlier prototypes of the system described here.

REFERENCES

- [1] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., Zukowski, D., “Challenges: An Application Model for Pervasive Computing”, To appear in the proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, Mobicom 2000.
- [2] Boyd J., “Floor control policies in multi-user applications”, in INTERACT ’93 and CHI ’93 Conference on Human Factors in Computing Systems, ACM Press, p. 107–108, 1993.
- [3] Brad A. Myers, et al. “Using Hand-Held Devices and PCs Together”. ACM Communications, 2001. To appear.
- [4] Brad A. Myers, Yu Shan A. Chuang, Marsha Tjandra, Mon-chu Chen, and Chun-Kwok Lee. “Floor Control in a Highly Collaborative Co-Located Task”.
- [5] Community Grids Lab (CGL), <http://communitygrids.iu.edu>
- [6] CPN Tools. CPN Tools Homepage. <http://wiki.daimi.au.dk/cpntools/>
- [7] Dommel H.P. and J.J. Garcia-Luna-Aceves, “Design issues for floor control protocols”, In Proceedings of SPIE Multimedia and Networking, (San Jose, CA, USA), pp. 305–16, February 1995.
- [8] Dommel H.P. and J.J. Garcia-Luna-Aceves, “Floor Control for Multimedia Conferencing and Collaboration”, ACM Multimedia Systems, Vol. 5, No. 1, January 1997.
- [9] Dommel H.P. and J.J. Garcia-Luna-Aceves, “Comparison of floor control protocols for collaborative multimedia environments,” In Proceedings of SPIE Symposium on Voice, Video, and Data Communications, (Boston, MA), November 1998.
- [10] D. Saha and A. Mukherjee. Pervasive Computing: A Paradigm for the 21st Century. Published by the IEEE Computer Society, Vol. 36, No. 3. pp. 25-31 March 2003.

- [11] Edwards, W.K. "Flexible Conflict Detection and Management in Collaborative Applications," in Proceedings UIST'97: ACM SIGGRAPH Symposium on User Interface Software and Technology. 1997. Banff, Alberta, Canada: pp. 139-148.
- [12] F. Berman, G. Fox, and A. Hey, editors. Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons, 2003.
- [13] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara. Global Multimedia Collaboration System in Proceedings of the 1st International Workshop on Middleware for Grid Computing co-located with Middleware 2003, June 17, 2003 Rio de Janeiro, Brazil.
- [14] G. Gallasch and L. M. Kristensen. Comms/CPN library. <http://wiki.daimi.au.dk/cpntools/>
- [15] G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. Proceedings of the CPN'2001 Workshop.
- [16] Global-MMCS (Global Multimedia Collaboration System). <http://www.globalmmcs.org>
- [17] Greenberg, S. "Sharing views and interactions with single-user applications," Proceedings of the ACM/IEEE Conference on Office Information Systems. 1990. Cambridge, MA: pp. 227-237.
- [18] Greenberg, S. "Personalizable groupware: Accommodating individual roles and group differences," In Proceedings of the ECSCW '91 European Conference of Computer Supported Cooperative Work. 1991. Amsterdam: Kluwer Academic Press. pp. 17-32.
- [19] Greenberg, S. and Marwood, D. "Real time groupware as a distributed system: Concurrency control and its effect on the interface," Proceedings of the ACM CSCW Conference on Computer Supported Cooperative Work, October 22-26, 1994. North Carolina, ACM Press.
- [20] I. Foster and C. Kesselman, editors. The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, 1998.
- [21] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 2001. 15(3): p. 200-222.
- [22] iPhone. <http://en.wikipedia.org/wiki/IPhone>
- [23] J.D. Ullman, Elements of ML Programming. Prentice-Hall, 1993.
- [24] Kangseok Kim. "A Framework for Synchronous and Ubiquitous Collaboration" Indiana University PhD September 2007.
- [25] Katrinis K., Parissidis G., and Plattner B., "Activity Sensing Floor Control in Multimedia Collaborative Applications", 10th International Conference on Distributed Multimedia Systems (DMS 2004).
- [26] K. Jensen, Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, vol. 1, Basic Concepts. Monographs in Theoretical Computer Sciences. Springer-Verlag, 1997.
- [27] Malpani Radhika and Lawrence A. Rowe, "Floor control for large-scale MBone seminars", in Proceedings of the Fifth ACM International Conference on Multimedia, ACM Press, p. 155-163, 1997.
- [28] Mark Roseman, Saul Greenberg. GROUPKIT: a groupware toolkit for building real-time conferencing applications. Proceedings of the 1992 ACM conference on Computer-supported cooperative work. Toronto, Ontario, Canada. 1992.
- [29] Matt Bishop. Introduction to Computer Security. Addison Wesley.
- [30] M. Handley, I. Wakeman, and J. Crowcroft, "CCCP: Conference Control Channel Protocol: A Scalable Base for Building Conference Control Applications," in ACM Conf. SIGCOMM., August 1995.
- [31] Moore's Law. http://en.wikipedia.org/wiki/Moore's_Law
- [32] M. Weiser. "The Computer for the Twenty-First Century," Scientific American, September 1991.
- [33] NaradaBrokering, <http://www.naradabrokering.org>
- [34] R, B'Far. Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML, Cambridge University Press 2005.
- [35] Scalable Vector Graphics (SVG). <http://www.w3.org/Graphics/SVG/>
- [36] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of the ACM/IFIP/USENIX Middleware Conference. 2003. pp 41-61.
- [37] Shrideep Pallickara, Harshawardhan Gadgil and Geoffrey Fox. On the Discovery of Topics in Distributed Publish/Subscribe systems Proceedings of the IEEE/ACM GRID 2005 Workshop, pp 25-32. November 13-14 2005 Seattle, WA.
- [38] Stefik, M., Bobrow, D.G., Foster, G., Lanning, S. and Tatar, D. (1987) "WYSIWIS revised: Early experiences with multiuser interfaces." ACM Transactions on Office Information Systems, 5(2), pp. 147-167, April.

- [39] SVGArena. <http://www.svgarena.org/>
- [40] Treo 600. http://en.wikipedia.org/wiki/Treo_600
- [41] Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Harun Altay. "Design and Implementation of a collaboration Web-services system", Special issue on Grid computing in Journal of Neural Parallel and Scientific Computations (NPSC), Volume 12, pages 391-408 (2004).
- [42] Wi-Fi (wireless fidelity). <http://en.wikipedia.org/wiki/Wi-Fi>
- [43] Xiaohong Qiu, Bryan Carpenter, Geoffrey Fox, Collaborative SVG as a Web Service, SVG Open 2003 Conference and Exhibition, Vancouver, Canada, July 2003.
- [44] Xiaohong Qiu. "Message-based MVC Architecture for Distributed and Desktop Applications" Syracuse University PhD March 2 2005.