Mais Hasan, Robbi Golle, Maik Debes, Jochen Seitz,
Agnieszka Lewandowska

Peer-to-peer collaboration in PeCoCC framework

Original published in:
International journal on advances in networks and services. - [S.l.] : IARIA. – 7 (2014), 3&4, p. 148-162.
ISSN: 1942-2644
URL: http://www.iariajournals.org/networks_and_services/tocv7n34.html
[Visited: 2015-01-27]

Peer-to-Peer Collaboration in PeCoCC Framework

Mais Hasan, Robbi Golle, Maik Debes, Jochen Seitz
Communication Networks Group
Technische Universität Ilmenau
Ilmenau, Germany
{first.last}@tu-ilmenau.de

Agnieszka Lewandowska
Staatliche Studienakademie Thüringen
Berufsakademie Eisenach
Eisenach, Germany
lewandowska@ba-eisenach.de

Abstract - **Computer-supported collaborative learning is an important domain of e-learning dealing with researching efficient methods to encourage people to learn together with the help of their computers. Many learning environments support this kind of collaboration and provide several applications to fulfill the needs for efficient learning. These learning environments are usually client-server based solutions, where the users have to access the responsible server to get the services and data they need. This fact leads to two main problems of scalability and single point of failure. Because of its structure, which is similar to the collaborative learning network, the peer-to-peer technology is suggested as a solution for these problems. This paper introduces a framework for using Peer-to-Peer communications, protocols and services in the area of computer-supported collaborative learning. It consists of four layers and includes different Peer-to-Peer modules, which will be selected according to the application requirements. It also provides a messaging system, which uses the overlay-multicasting protocols of the supported Peer-to-Peer modules. This paper shows the advantage of the proposed framework.**

Keywords - Peer-to-Peer communications; computer-supported collaborative learning; Scribe overlay multicasting protocol.

I. INTRODUCTION

During the last decade, online learning has gained enormous interest in most educational institutes. E-learning can be defined as the process of using electronic media, information and communication technologies in education. E-learning includes numerous forms of educational technology in learning and teaching and can be used jointly with conventional face-to-face learning. E-learning can occur in or out of the classroom. It is suited to distance and flexible learning and can be asynchronous or synchronous. As a result of the rapid improvement in the areas of education, information and communication technologies, various e-learning methods have evolved. This evolution started with using the information technology in Computer Based Training (CBT) and developed in the direction of exploiting the Internet and social interaction in Virtual Learning Environments (VLE) and Computer-Supported Collaborative Learning (CSCL) applications.

The most used learning environments have been based on the client/server approach. A server is the source of services and information, several clients have access to. However, the approach suffers from two main problems: scalability and single point of failure. Thus, different approaches have been developed to overcome these problems. One of these is a paradigm shift to the Peer-to-Peer (P2P) model, which offers better load balancing and robustness compared with the conventional client-server model. In this approach, the communication partners act as servers and clients at the same time. They all offer a part of the information and retrieve information from other nodes known as peers. The more peers take part in a P2P network, the better this network scales and the higher its reliability is. Several application fields have utilized this P2P approach so far. In this paper, we introduce a framework to apply this approach to computer-supported collaborative learning.

Therefore, the paper is organized as follows: Section II shortly deals with computer-supported collaborative learning and reviews some CSCL-tools based on P2P technology. Different P2P technologies and their properties are analyzed in Section III. Section IV presents our designed CSCL-tool; the **pe**er-to-peer **co**mmunications for **c**omputer-supported **c**ollaborative learning (PeCoCC) framework and its functioning [1]. A description of the used software and a primary implementation of Scribe overlay multicasting protocol in the P2P simulator PeerfactSim.KOM follows in Section V. Section VI gives an overview of the current state of the work and summarizes the paper.

II. PEER-TO-PEER COMPUTER-SUPPORTED COLLABORATIVE LEARNING

Computer-supported collaborative learning is an emerging branch of e-learning allowing several students to cooperate with each other and with the teaching staff online in order to solve shared tasks or to exchange their skills. Computer-supported collaborative learning is related to collaborative learning and Computer Supported Cooperative Work (CSCW). By collaborative learning, we generally mean that a group of students work together to discuss, solve or evaluate teaching materials; on the other hand, computer supported cooperative work addresses the technologies and tools supporting people in their work. Hence, computer supported collaborative learning refers to the use of CSCW-technologies and tools by a group of collaborative students in a learning process. These technologies and tools have been developed to provide an efficient learning process. Woodill [2] gives an overview of

all the different technologies used to support collaborative learning (see Fig. 1).

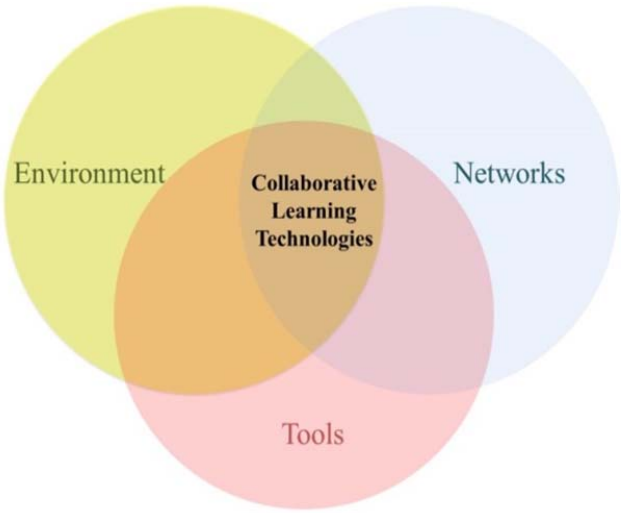

Figure 1. Information technologies used to support collaborative learning [2]

Generally, collaboration among the students is accomplished by many scenarios, which offer team work and the distribution of teaching materials among the users themselves. People participating in such teams need to be able to access the offered resources and services regardless of their current location. They also need effective means to communicate and cooperate with each other. In general, we can recognize three collaborative scenarios according to three attributes, the time, the place and the tasks of the collaboration. The first scenario can be called ad-hoc scenario, which provides collaboration in a limited geographical area and enables the users to interact and communicate with limited collaborative functionalities. Typical examples are conferences where people meet, exchange contacts and ideas, but do not work together on a shared task. The second possible cooperative scenario is the short-term collaboration, which covers a limited time period and enables the group members to collaborate in order to achieve a shared task, e.g., work on a paper, or integrate software components. Such collaboration requires access to shared resources as well as knowledge and data exchange. It is also based on a trust relationship among the members, who do not have to be in the same place. The third scenario presents the long-term collaboration, which is used by teams for longer time periods, e.g., within a project. In this case, it is necessary to offer flexible and different interaction and cooperation possibilities and services. Usually, a centralized infrastructure is used for this scenario.

In all these scenarios, the user must be able to find and access the required resources and services. In this case an efficient distributed search based on the semantic description of the according data, users, services, etc. is needed. Moreover, distributed storage and a secure access to the resources are really important in these collaborative systems. Furthermore, a team membership management service, which should be associated with a trust model as well as a cooperative messaging scheme among the users, is necessary to provide a flexible interaction among the group members [3].

P2P systems as one of the technologies, which are used in the collaborative systems, provide the efficient, scalable distributed search, while the semantic search in P2P networks embodies one of the important current research areas. Some more research work has been done to provide distributed storage and cooperative messaging to some P2P technologies. Furthermore, educational P2P applications like, e.g., COMTELLA, EDUCOSM, Edutella, and Groove have been developed for some specific needs and they are still under development. A brief description with a comparison between these applications will be highlighted in the following.

COMTELLA is a P2P file sharing system that allows students to contribute and share class-related resources with their community [4]. The shared papers are annotated with respect to their content in categories. COMTELLA uses a modified version of the standard Gnutella P2P protocol and instead of sharing the actual files, only their URLs are shared. A list of the shared articles, their URLs in the web and the corresponding comments are distributed among the users. There is one list for every category. If a student searches for a paper, he should only search the list of the matching category. Students can view and read the papers without downloading them by clicking on the “Visit” button in the COMTELLA user interface, which starts the default browser with the URL of the paper.

EDUCOSM is a web-based learning environment providing a shared view to the Web [5]. It consists of a collection of server-side scripts and an HTML and JavaScript based client that runs inside a web browser. The role of the server is to store the data and act as a proxy between the client and the rest of the web. The principles of EDUCOSM and COMTELLA are similar with the difference that the storage of the data in COMTELLA is distributed among the users.

Edutella is an educational P2P network built on Sun Microsystems JXTA Framework [6]. Edutella is an open source P2P application for searching semantic Web metadata. It uses the resource description framework (RDF) for presenting information in the web. Edutella deals with metadata about content, not with content itself. It adds a search service to the JXTA framework, so that any node that carries metadata about some resource can announce an Edutella search service to the network. The nodes in Edutella have at least one of three types of roles: provider (provides a query service), consumer (asks questions) and hub (manages query routing in the network).

The previously mentioned three P2P collaborative systems offer only one collaborative tool, mostly a file-sharing application, which is not sufficient for efficient collaboration among the users. These systems suffer the

absence of a coordinative tool like a group calendar, which is typical for team or group software. They also do not support cooperation applications like a whiteboard or a text editor.

These problems have been tried to be solved in one of the popular collaborative environments, Groove. It is a collaborative groupware based on the principle of a shared workspace [7]. Tools like a shared browser, a shared drawing board or a file archive are used to operate in this shared workspace. Groove provides servers that are used to detect new peers in the network and to store content if one or more peers are offline and cannot see the changes made at that time. Using server-based services threatens the availability of these services if one of these servers fails.

Groove is targeted at small workgroups and has its own protocols. It is only available for the windows platform, so it suffers interoperability problems. Table I presents a brief comparison between the previously mentioned collaborative systems.

TABLE I. A COMPARISON BETWEEN SOME P2P COLLABORATIVE SYSTEMS

P2P based CSCL Applications	Pure P2P Connectivity and Services	Collaborative Applications	Coordinat-ion and Awareness	Interoper-ability
COMTELLA	+	o	-	+
EDUCOSM	-	o	-	+
Edutella	+	o	-	+
Groove	-	+	+	-

The comparison is based on four attributes, which reflect the flexibility and the effectivity of any P2P collaborative application. The first attribute represents the usage of pure P2P communication and services, which is supported by COMTELLA and Edutella, while the other systems need the functions of servers to provide some services like saving the profiles of users or offering the collaboration space awareness.

The second attribute embodies the effective support of collaborative applications, which is performed very well in Groove, while the other systems offer only one application like a file sharing application or a semantic search application, which is not enough for an efficient collaboration among the users. Unlike Groove, the other three systems do not support any collaboration space awareness or coordination among the users, which is an important attribute to improve the efficiency of the collaboration. The fourth attribute is the availability for different operating systems or in other words the interoperability, which is considered only in COMTELLA, ADUCOSM and Edutella.

The previous comparison manifests the need for a collaborative platform providing many collaborative and coordinative tools, supporting interoperability and work space awareness, and basing on fully distributed server-independent P2P communications and services. However, there is no open source software having the mentioned functionalities available at the moment.

To be able to understand the need of the previously mentioned structures for efficient using of P2P networks in a collaborative scenario, an overview on the P2P technologies with some examples is highlighted in the following section.

III. PEER-TO-PEER TECHNOLOGIES

In contrary to the client-server model, all the members of a P2P network are equally offering and requesting services. Generally, we can assert that every P2P network is established on an overlay network, mostly based on Transmission Control Protocol (TCP) or on Hypertext Transfer Protocol (HTTP) connections. Thus, the overlay and the physical network can be separated completely from each other. Hence, the overlay connections do not reflect the physical connections. Nevertheless, it is possible to match the overlay to the physical network if necessary. P2P networks can be divided into two classes: structured and unstructured P2P networks.

A. Structured P2P networks

In a structured P2P network, the network topology and the location of content is determined by employing a P2P protocol. In these networks, the content and the participating nodes share the same address space, which makes it easy and expeditious to reach any content in this space. Structured P2P networks are based on a Distributed Hash Table (DHT) and have no central entities. Frequent signaling traffic is necessary to maintain the network awareness of the nodes [8]. Pastry, Chord and Content Addressable Network (CAN) are examples for this class, two of them, which are used in the PeCoCC framework will be briefly highlighted in the following.

Pastry is a distributed hash table (DHT) algorithm that empowers Internet nodes to build a structured P2P overlay network. It serves as a basis to build up distributed network applications on the top of it [9]. Additionally to its IP address, a Pastry node possesses a unique 128-bit nodeId. It is randomly assigned from a circular nodeId space, ranging from 0 to 2^{128}-1. The assignment is performed in a way, so that nodeIds are uniformly distributed among the nodeId space. As a result, nodes with adjacent nodeIds differ in characteristics such as geography, network attachment or ownership. Routing in Pastry is performed with the help of key values. A corresponding command looks as follows: route (msg, key). A message msg will be routed to the (still alive) node with nodeId numerically closest to the key value. Assuming N nodes are alive within a network, Pastry routes as described within $log_{2^b}(N)$ steps. Delivery of a message is guaranteed when less than $L/2$ nodes fail at the same time. L and b are configuration parameters, where typically $b = 4$ and $L = 16$ or 32. Every key and nodeId in Pastry is a sequence of digits with base 2^b. During a route step, a pastry node routes the message to another node whose nodeId has a prefix, which is one digit ($2^b bits$) longer than the prefix the key shares with the current node´s

nodeId. If there is no such node, the message will be forwarded to a node, whose nodeId shares a prefix with the key as long as the current node´s, but numerically closer to the key than the nodeId of the current node. If still there is no such node, then the message has arrived at its destination. Fig. 2 shows a simple routing example [10].

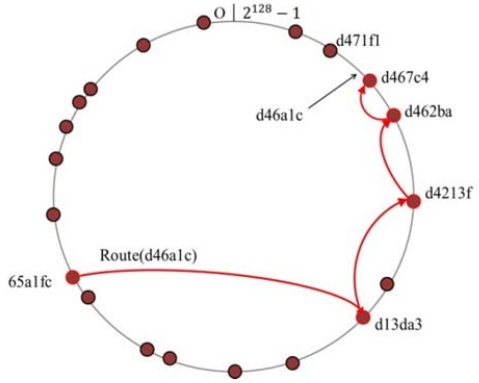

Figure 2. Message routing in Pastry nodeId space [10]

The routing tables of Pastry nodes contain $(2^b-1)* log_{2^b}(N)$ entries. Each entry maps a nodeId to the associated node´s IP address. In the case of node arrival, departure or failure, the routing tables needs to be updated. That can effectively be done by exchanging $O(log_{2^b}(N))$ messages. In addition to the routing table, a node supervises a leaf set. In this leaf set, the node stores the $L/2$ numerically closest larger and smaller nodeIds. Within any network type, inconsistences, like node´s joining, leaving or failing, happen. These inconsistences have to be considered by maintaining the routing table and leaf set of nodes. Special messages are exchanged when some of these events happen, so that the nodes of a pastry network maintain up-to-date routing tables and leaf sets.

Since the PeCoCC framework uses Scribe, which is one of the important Pastry applications, a short overview is introduced in the following paragraph.

Scribe has been defined by its developers as a "scalable application-level multicast infrastructure built on the top of Pastry" [11]. With its help, Pastry nodes can create groups and exchange multicast messages among group members. It is possible to create, join and leave multiple groups. Scribe is able to maintain large amounts of groups with possibly large amounts of group members.

If a node wants to create a group, it determines a group name string, which is mostly the topic of the shared task of the group, and calculates a hash value from it. The groupID is created by concatenating the hash value of the group name and the nodeId (which already is a hash value). The creating node may furthermore decide on group credentials which determine the node characteristics that have to be fulfilled in order to allow this node to join the group. The create message will be sent using the route method of Pastry, where the key in this case is the groupID. The create message will be delivered to the node with nodeId numerically closest to the groupID. This node that is now called the root of the group, then adds the groupID to a list of groups it already knows about.

If a node wants to join a group, it routes a join message through the network. The key of the route method is in this case also the groupID of the group the node wants to join. At each hop in the network, Scribe´s forward method will be invoked. If a node receives the join message, it will look up its groups table if it already knows the group. If not, it adds the group to its groups table. It also terminates the join message and routes a new one on its own, again, with the groupID as a key. Furthermore, it adds the nodeId of the sending node to its children list of the group. This procedure is repeated at the next hops until the message arrives at the root of the group, which in turn adds the nodeId of the last node to its children table of the group.

By terminating the join message and routing a new one at each hop, all nodes will store only the nodeId of the previous nodes in their children lists of groups. This behavior will create a tree-like structure to which multicast messages can be delivered.

If a node wants to distribute a multicast message to a certain group, it routes the message to the root of the group. The message will arrive at the root, which will first deliver its IP address to the sender of the multicast message. In this way, the sender can use the IP address of the root to directly send multicast messages to it in the future, which saves repeated routing. Now, the root sends the message to all the children of the group it is aware of (which are only one hop away). These children will then route the message to the children of the group they are aware of. This behavior is repeated until the message arrives at the leaves of the multicast tree. These leaves are the members of the group. The nodes along the multicast tree are called forwarders of the group, as they forward multicast messages through the tree. Forwarders can also be members of the group. Fig. 3 illustrates a multicast group, which has a root with nodeId (1100) that is numerically closest to the groupID (in this case 1100 too) [12].

If a node wants to leave a group, it locally deletes its nodeId from its children table of the group. If the children table of that group is empty, it sends a leave message to its parent in the multicast tree (the next hop of a route message). The parent deletes the sender's nodeId from its children table of the group. If the children table is empty, the parent sends a message to its parent nodes. This behavior is repeated until the leave message arrives at a node, which has more than one entry in the according children table.

The second used P2P overlay is the Content Addressable Network (CAN), which is also a structured P2P network based on the distributed hash table concept [13]. The key space in CAN is organized as a virtual d-dimensional torus with Cartesian coordinates. This coordinate space is completely logical and has no relation to any physical

coordinate system. It is partitioned among the participating nodes, so each node is responsible for an area of key identifier space, called a zone. This node also maintains information like a coordinate routing table with IP addresses and a virtual coordinate zone of each of its adjacent neighbors in the coordinate space.

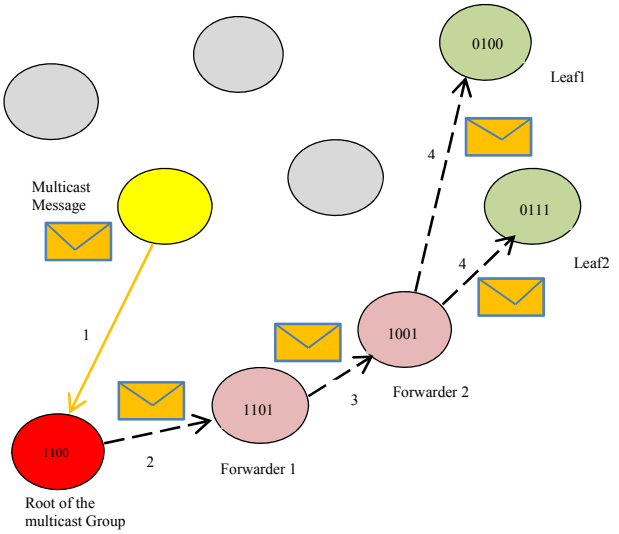

Figure 3. Sending a multicast message to the group with a groupID (1100)

The coordinate space in CAN is used to save (*key*, *value*) pairs by mapping the key onto a point *P* in the coordinate system using a uniform hash function. The corresponding (*key*, *value*) pair is then stored at the responsible node of the zone within which the point *P* lies. To retrieve any entry relevant to any key, any node can apply the same hash function to map that key onto point *P* and then retrieve the corresponding value from the responsible node of the point *P*. A CAN message includes the destination coordinates. Using its neighbor coordinate set in its routing table, a node forwards a message to the neighbor with coordinates closest to the destination coordinates. For a *d* dimensional space divided into *n* zones, the average routing path length is $(d/4)*(n^{1/d})$ hops and every node maintains $2d$ neighbors. Taking into account that many different paths exist between two points in the space, a node can automatically route along the next best available path, if one or more of its neighbors failed. One method to improve the performance of CAN includes using multiple hash functions to map a single key to several points of the coordinate space and hence increase the availability. Another method to reduce the routing delay in CAN is maintaining several independent coordinate spaces called realities.

B. *Unstructured P2P networks*

The distribution of nodes and content in unstructured P2P networks is executed randomly. The position of content can only be resolved in a local proximity of a node and only by flooding the request to a particular extent. In this way, these networks consume the bandwidth, which has been saved by their random distribution. Unstructured P2P networks can be centralized with an index server like Napster, hybrid with dynamic super nodes like Gnutella 0.6 and JXTA, or pure without any central entities like Gnutella 0.4 and Freenet [8]. The following paragraph gives a brief overview of the Gnutella 0.4 protocol, which supports pure P2P connection and presents the simplest form of P2P function in unstructured networks.

Gnutella 0.4 is a decentralized Peer-to-Peer file sharing protocol. Every node in a Gnutella network performs the tasks of client and server at the same time. For this reason these nodes are referred to as servents and the Gnutella protocol defines the way in which these servents communicate over the network. The Gnutella protocol uses ping message flooding to discover hosts on the network. If a servent receives a ping message, it has to respond with one or more pong messages, which include the address of a connected Gnutella servent and information regarding the amount of available data it is sharing on the network. Once the address of a connected servent on the network is obtained, a TCP/IP connection to this servent is created. Thereby, a new servent will be able to take part in a Gnutella network. If one of the servents wants to search for file-based data in the network, it sends a query with some search criteria to its neighbors, which forward this query to their neighbors too, if they do not have the researched file. The forwarding of the query will be continued till the file-based data is found or the time to live field (*TTL*) in the query, which will be decreased by every hop, is found to be zero. If a servent has the research data, it responds with a QueryHit message, which contains enough information to acquire the data matching the corresponding query. This information includes the IP address and the Servent Identifier of the responding host as well as the port number on which the responding host can accept incoming connections and the result set, which identifies file name, file size and file index in the responding host [14].

The servent that receives the queryHit as an answer of its query will initiate a direct download of the requested file, i.e., a direct connection between the source and the target servents is established in order to perform the data transfer. To solve the problem that can arise when the responding servent is situated behind a firewall blocking incoming connections to its Gnutella Port, the servent, which received the queryHit sends then a push message to the servent that sent the queryHit and is supposed to have the researched file. The receiver of the Push message should try to establish a connection to the requesting servent, identified by IP address and port number included in the push message. If the requesting servent is behind a firewall, a connection in the context of Gnutella protocol is not possible.

IV. THE PECOCC FRAMEWORK

The important properties of P2P technologies make them an attractive searching approach to improve performance of

some collaborative systems. Until now, most P2P collaborative environments are developed for specific needs and a central entity is used in most of them. Therefore, we have developed a P2P framework for computer-supported collaborative learning, which we called PeCoCC. The PeCoCC framework has been designed to be used in short-term scenario, which serves a limited number of users within a limited time period. It uses different P2P overlays to support different applications. This characteristic of the PeCoCC framework enables completely separate working of the applications, which increases the robustness of the system. The PeCoCC framework has a layered architecture depicted in Fig. 4 [1].

According to the requirements of an efficient collaboration in the short-term scenario, the main services that this framework has to provide are as follows:

Collaborative Tools – The PeCoCC framework provides three applications, which are important to cooperate efficiently. A shared calendar will be used to allow the users to organize their regular meetings; a distributed text editor can be used to jointly make notes on a given subject or to brainstorm about a topic and P2P file sharing allows users to access the distributed contents they need to cooperate.

Session Management Service – Participants in a particular application session, e.g., text editor session, have to be able to get a list of the other members in this session. In this case, a session management unit in the application layer saves a profiles-list of the registered participants for every provided application. This list contains the UserIDs as well as the registered role and priority of every user in the corresponding application. A session management unit also maintains the last case of the application and implements an "Initiation" mechanism to consistently provide the information for latecomers to enable them to participate in the ongoing session. This is typically achieved by getting the state of the distributed application from the current participants and by initializing the application of the latecomer with this information.

Membership Management Service – To be able to use the framework, the user has to be identified by the membership management unit. This unit has to be associated with a trust model, which provides every registered user with an application-ID number (APP-ID number). This number identifies the message traffic in the framework, so that only the registered user can send legal messages.

Repair Mechanism – If a participant in a session suddenly or with intent is going offline, or if the connection of a session member is broken because of technical problems, this mechanism has the task to repair the overlay topology of the network, so that the ongoing session should not be disrupted. This mechanism covers also a late join service, which maintains the overlay topology in the case of newcomers.

Synchronization – For some application modules (e.g., distributed text editor), the group members need to be synchronized to interpret the events in the correct time and order. Furthermore, the most repair algorithms are based on the estimate of the network delay.

Security – As it is mentioned previously, the PeCoCC framework provides many collaborative tools, which based on the exchange of knowledge and data. Therefore, a reliable security mechanism, which controls the participation of a session and takes into account the individuality of the data and documents, is needed. Also, the framework provides security mechanisms (e.g., encryption) to keep personal data secure.

Group Management – Beside the membership management and the session management services, the framework must include functions to manage the communication among collaborative group members in the overlay network. These functions are achieved in the PeCoCC framework using the concepts of overlay multicasting and the cooperative messaging scheme among the peers in the overlay topology.

PeCoCC Messaging System – The PeCoCC framework uses its own message scheme and identifies thereby its used massages in the network and between the layers of the framework. Having an own messaging scheme makes it possible for one user to be member in more than one session and one application. It helps by connection between two overlays, which a user is member in.

The PeCoCC framework also presents the concept of combination between a specific application and the most appropriate P2P overlay for this application. This concept will improve the robustness of the overall system and secure the availability of the other applications in case of attack of one overlay by some possible malicious peers.

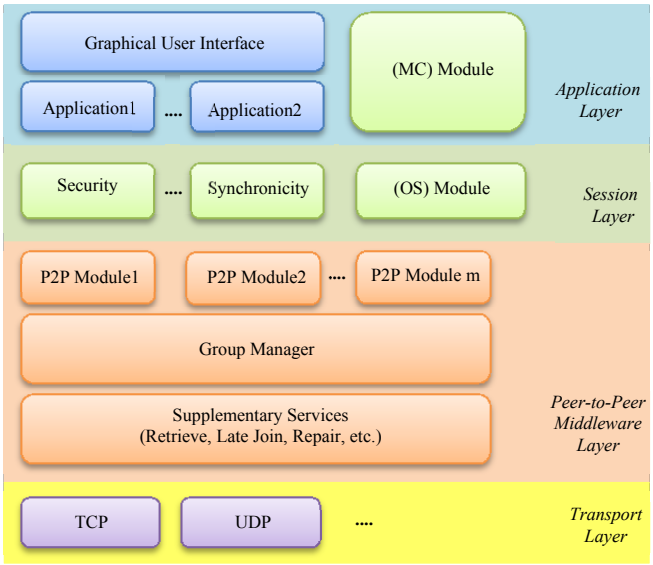

Figure 4. The PeCoCC framework [1]

Therefore, the PeCoCC framework consists of four different layers, which provide its main functionalities and services. The layers are introduced in the next subsections.

A. Application Layer

The application layer consists of three main parts. The graphical user interface allows the interaction between the user and the framework. It facilitates a consistent operation of all the desired CSCL services. The PeCoCC Management and Control (MC) module is responsible for controlling the data flow through the framework and the work flow among the users. It embodies the user's membership management and the session management services in the PeCoCC framework. The MC module influences the application program and also saves a list of the cooperating participants, their application-dependent roles and their priorities. The peers should be identified by the MC module to be allowed to enter the system. The rights of the users can be defined by the applications themselves. In a file sharing application, for example, all the users have equal rights, while in a text editor; the teacher should have more possibilities to manipulate the entered information than the students. Furthermore, in sessions without a dedicated chair, the MC module is responsible for defining the user that has the according rights of a chair.

The third part is composed of application modules. These can be freely chosen and added on demand. As stated above, the first modules of our choice are a distributed text editor application, a calendar, and a file sharing application. These modules interact with the graphical user interface and the MC module.

B. Session Layer

The session layer provides general mechanisms that are necessary for the offered applications. Currently, we have concentrated on two mechanisms. The security module is responsible for securing private user data and the synchronicity module provides a mechanism to synchronize the different group members so that all of these receive the events in the same order. This service is necessary for real time applications like a distributed text editor. Since the usage of a P2P overlay in the PeCoCC framework is application dependent, the session layer includes the Overlay Selection (OS) module, which is responsible for saving the information about the appropriate P2P overlay for each application. This information will be obtained by several experiments according the latency and load of the network taking into account the requirements of each application.

C. Peer-to-Peer Middleware Layer

As P2P technologies exist with respective advantages and disadvantages, the PeCoCC framework allows the usage of different pure P2P technologies. Each technology is encapsulated in a P2P module and offers its communication functionality. Which module should be used is selected by the OS module in the session layer according to the needs of the application. To illustrate the functionality of this layer, we have started with two well-known P2P approaches.

The first overlay is the content addressable network (CAN), which can be used for an efficient distribution of information and teaching materials. As it is mentioned in Section III, it is a structured P2P network based on DHT. CAN offers high scalability and reliability and provides more load balancing than any other pure P2P overlay [15], but it does not take into account the underlying network conditions. Therefore, it is not suitable for real-time applications due to the fact that it does not make any correlation between the overlay distance and the actual number of unicast hops between the hosts in the underlying network.

The second overlay is the structured P2P network Pastry, which considers the underlying network topology and supports a scalable and distributed object location and routing in application layer [8]. Pastry provides an overlay multicasting protocol and collaborative messaging scheme, thus it can be integrated in a P2P module for applications like a distributed text editor and instant messaging.

The Peer-to-Peer Middleware layer also comprises a set of supplementary services that extend the P2P modules with late join, retrieval and repair functions

Furthermore, in each overlay network, the users belonging to one user group have to be managed. This is done in the module called group manager. This module is responsible for forming and supervising a collaborative user group in the overlay network. It uses the concept of overlay multicasting to achieve these tasks.

Overlay multicast implements a multicast service at the overlay network layer. Peers participating in a multicast session use the routing and messaging structure of this overlay. Overlay multicast is achieved through message forwarding among the members of the multicast group at the overlay network using unicast across the underlying network or Internet. So the hosts in overlay multicast handle group management, routing, and tree construction, without any support from the conventional Internet routers, which makes overlay multicasting much easier to deploy than IP multicasting [16].

As Pastry and CAN have been selected as first two P2P overlays in the PeCoCC framework, two overlay multicast protocols built on the top of these overlays have been investigated. CAN-based multicast uses the flooding of messages to all the participating nodes in the group. The members of the multicast group first form a group-specific "mini" CAN and then multicasting will be achieved by flooding over this mini CAN [17]. Scribe is a tree based overlay multicast mechanism built on top of Pastry. It builds a single multicast tree for the whole group. Every tree has a root that is responsible for distributing the message among the multicast tree, which is created by combining Pastry paths from each group member to the tree root [11]. The implementation of our work has been started with Scribe protocol, which is described in more detail in the following section.

D. Transport Layer

The transport layer embodies the known transport layer in the OSI model, which provides access to different commonly used transport protocols. In PeCoCC framework, an appropriate transport protocol will be selected accordingly to the requirements of the opened applications. For example, the distributed text editor utilizes the Transmission Control Protocol TCP, which provides a reliable transport service.

E. Functioning of the System

To understand the functioning of the PeCoCC framework, a collaborative scenario will be presented in the following.

A user starts an application and wants to create a collaborative session with other users, who he knows. The peer then, which is used by this user, will send an announcement with a defined groupID, which can be a hash value of the topic of the session, in the network. In this way, the other peers will be able to find it and join the collaborative session or group. The users how join a collaborative group, receive from one of the group members the all information, which is needed to be able to interact in the ongoing session (e.g., UserIDs of the other group members, the current state of the opened application and its corresponding data, etc.). Every user has its role and priority, which are registered in its profile in the framework. Only the user with an administrator role can change these registered data. According to this scenario, the functioning of the framework will be highlighted in the following.

When the user starts one of the available applications (e.g., a distributed text editor), the MC module is activated and sends a CHOOSE message to the OS module in the session layer containing information about the opened application. The OS module then decides on the basis of the opened application which P2P module is more appropriate for this application and replies to the MC module in the application layer with an OVERLAY message. The OS module also activates the necessary services for the opened application.

The MC module receives the OVERLAY message and retrieves the saved list of the expected participants (the participants of an application should be previously registered by the MC module and saved in a specific list). The MC module then sends a START message to the corresponding P2P module in P2P middleware layer. In the P2P middleware layer, the selected module starts the P2P connection and takes part in the P2P network. The group manager sends a DISCOVERY message with a group ID to find out if the collaborative group with group ID in the P2P network has already been built or not. If it receives a positive answer, the group manager then sends a JOIN message with the group ID using the overlay multicasting protocol to join a collaborative group and retrieve the information about the participating peers as well as the important data to interact in the current session. This case is illustrated in Fig. 5.

The information retrieved by the group manager is returned to the MC module to specify the role and the rights of the peer in the session. The role of the peer will appear in form of active or inactive interaction possibilities in the user interface.

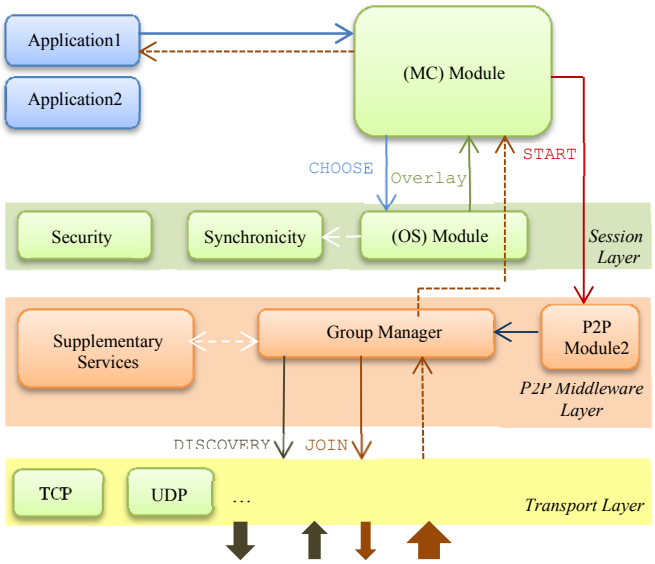

Figure 5. Joining the already built session group

If a collaborative group does not exist, the group manager will send a CREATE message in the P2P network to create a collaborative group with a specific group ID. It will also wait for any join message sent by other peers, which want to join the group. Fig. 6 shows the whole process in this case.

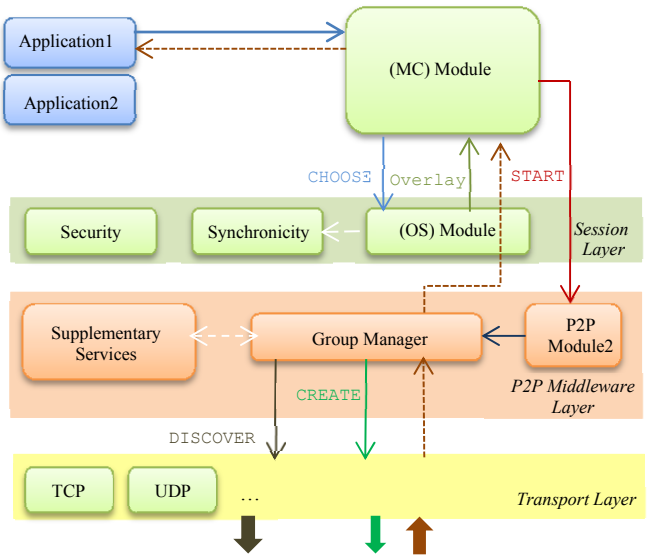

Figure 6. Creating a new session group

Which rights and roles the user has, is managed in tables and saved in the MC module. One table is defined for each

application. In the case where all users have the same rights, a mechanism to manage and identify the roles will be used.

This mechanism can take into account the registering sequence of the participants or the alphabetic order of the user ID names, etc.

V. SOFTWARE AND IMPLEMENTATION

The PeCoCC framework is still in implementation phase. It supports the interoperability and is being implemented in Java to be independent from the underlying platform. Three P2P simulators have been studied to select the most suitable one, which cover the main structures of our framework. These simulators are FreePastry [18], PeerSim [19] and PeerfactSim.KOM [20]. These three simulators have been chosen because they are popular among the P2P research community. Furthermore, all three are implemented in Java, the programing language of our framework. In this work, the PeerfactSim.KOM simulator is used because of its visualization capabilities and detailed documentation, which are not available in the other both simulators. In this section, we give a brief overview of the used PeerfactSim.KOM simulator and present the implementation of one of the missing structures in this simulator, which is necessary for the evaluation process of our framework.

A. PeerfactSim.KOM

PeerfactSim.KOM is a stand-alone simulator created for simulating P2P Networks. It is a purely event-based simulator and has a layered architecture. This layered architecture is similar to the layered structure of the PeCoCC framework and comprises an application layer, a service layer, an overlay layer, a transport layer and a network layer. At the network layer churn, jitter and latency can be modeled and adjusted. The transport layer currently supports UDP and TCP. The overlay layer contains the different P2P overlay protocols, which PeerfactSim.KOM is capable of simulating. The service layer offers additional services, the most important one is monitoring. The application layer hosts P2P applications. Currently, only a file sharing application is implemented [20].

PeerfactSim.KOM simulator as mentioned above provides implementation of numerous P2P protocols like Pastry, CAN, Gnutella, etc. It also offers a user-friendly graphical user interface (Fig. 7).

For every P2P protocol, that shall be simulated, there is an XML-based configuration file, which specifies the used layers, the P2P protocol, and the maximum number of nodes, as well as the maximum simulation time and the classes for data collection.

For gathering data arising from the simulation, the PeerfactSim.KOM simulator offers logging and statistic functionality. The logging is responsible for recording the events during a simulation like nodes joining and failing. The statistic functionality gathers a considerable amount of statistical data from the simulation, such as number of failed messages, sent messages and received messages, as well as data from every participating node, such as number of neighbors, number of leafs, messages sent per second, etc.

Together with the considerable amount of gathered information, the simulator is able to give a visualization of the simulation with rich data in the GUI. Fig. 8 shows a screenshot of the visualization´s main window in case of Pastry-simulation.

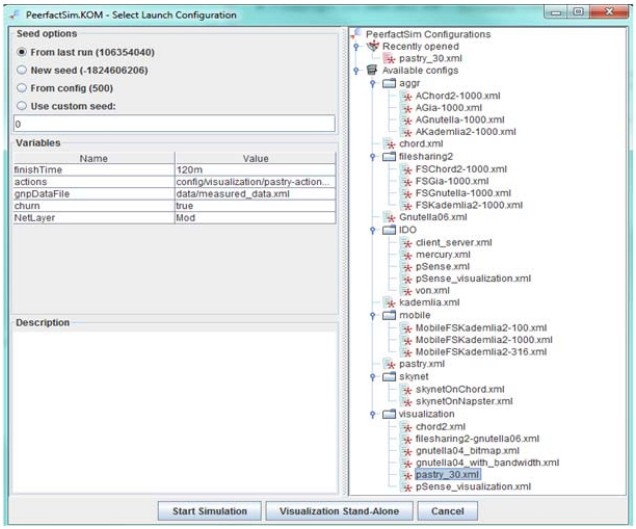

Figure 7. PeerfactSim.KOM ´s Graphical user interface

After the simulation is finished, a window is opened, showing an interface where the simulation can be reviewed in a media player-like fashion. A graphical view displays the nodes and their inter-network connections.

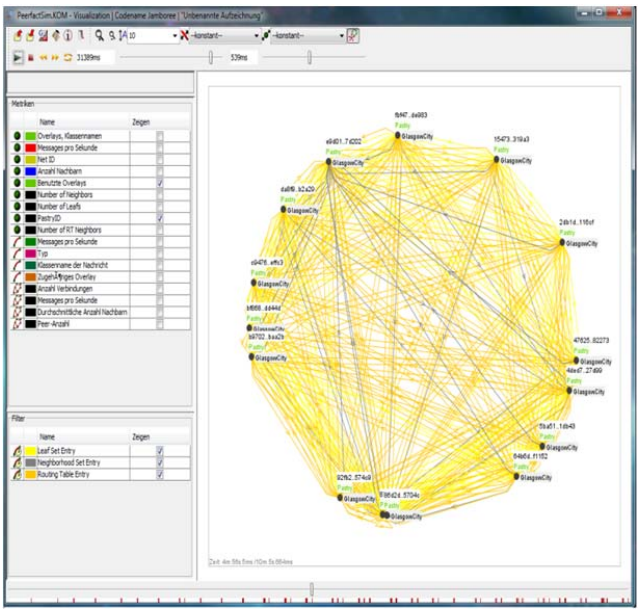

Figure 8. Visualization´s main window in case of pastry simulation

Various parameters can be viewed and displayed for the nodes and their connections like nodeId, number of neighbors, type of messages, etc. The time variant process can be started, stopped, paused, fast forwarded and rewound. The execution speed can be slowed and accelerated. Another valuable feature of the simulator is the Gnuplot-Export. The previously mentioned parameters can be plotted with the help of Gnuplot and displayed as functions over simulation time. A convenient visualization can be achieved this way.

PeerfactSim.KOM simulator offers the main block for testing and evaluating the functioning of the PeCoCC framework. It can be run in the eclipse environment and extended with more applications and protocols, which present the work of PeCoCC framework. As it is mention in the previous section, the Pastry overlay has been chosen to serve some real time applications like a distributed text editor and a shared calendar. The PeerfactSim.KOM simulator provides an implementation of the Pastry overlay but lacks some more important functions, which are known as applications of pastry itself. These functions are needed to fulfill the requirements of running real time applications on the top of the Pastry overlay. They comprise SCRIBE, the overlay multicasting protocol as well as POST, the co-operative messaging system, which extent the functionality of Pastry algorithm.

B. Pastry implementation in PeerfactSim.KOM

As one of DHT algorithms, Pastry is implemented in the package `impl/overlay/dht/pastry` at overlay network layer in PeerfactSim.KOM simulator. The implementation consists of many sub packages that define the components, messages and operations of Pastry. Fig. 9 shows a hierarchy presentation of the implementation of Pastry´s sub packages in PeerfactSim.KOM simulator. Some important classes in these sub packages will be described briefly in the following paragraphs.

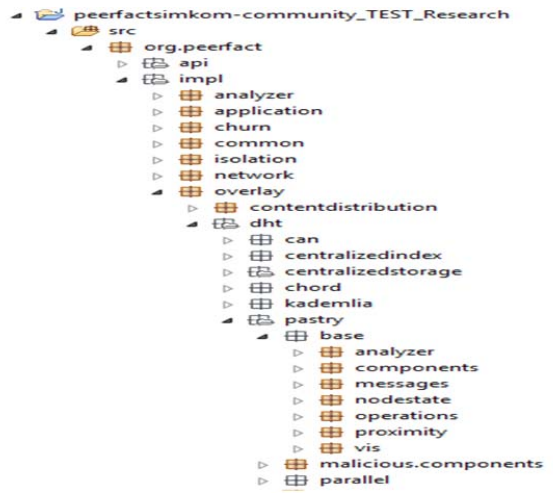

Figure 9. Pastry implementation in PeerfactSim.KOM simulator

Many important components of the Pastry overlay network are declared in the sub package `pastry/base/component`. For example, the class `PastryID` represents the overlay ID used by Pastry. Also the class `PastryNode` is used to represent node of the Pastry overlay and how it manages its leaf and neighbor sets as well as routing table. The method `route(msg, key)` is also declared in this class. The class `PastryMessageHandler` handles incoming messages and dispatches or processes them. The method `send(msg, IP)` is declared in this class.

All the messages of Pastry overlay are declared in the sub `package pastry/base/messages`. For example, the class `LookupMsg` is used to find the responsible node for a given target ID. Also the class `JoinMsg` represents the message sent by a node to join the pastry overlay.

In the sub package `pastry/base/operations` many operations of the pastry overlay are declared; e.g., look up and join operations. The following paragraph presents the primary implementation of Scribe in PeerfactSim.KOM simulator, which is still under development.

C. SCRIBE

As it is mentioned in Section III, Scribe is built on the top of Pastry; it heavily relies on its functionality. With the help of Pastry´s methods `route(msg, key)` and `send(msg, IP)`, the four messages types of Scribe can be distributed in the network. These messages are

- `create(credentials, nodeId)`,
- `join(credentials, nodeId)`,
- `leave(credentials, nodeId)` and
- `multicast(credentials, nodeId, message)`.

All Scribe messages use the `route` or `send` method of Pastry, where the key is the according groupID of the message, which will be described later. Scribe also has its two own methods, which instruct the nodes of the network how to handle Scribe messages. These methods are `forward` and the `deliver`.

In order to implement application specific behavior of Scribe, changes and extensions of the code have been performed on the application layer and overlay layer of the simulator. On the application layer, the behavior of the application encountering a certain event of the simulation is defined. On the overlay layer, the pastry implementation has been adapted to process and react to Scribe application behavior. Furthermore, a scenario triggering certain application behavior (called actions) has been defined in a file, as well as the simulator configuration in a configuration file.

1) Adaption of the Application Layer: The application has been divided into three functional units ensuring the desired behavior. The core of the Scribe application, which

comprises the first functional unit, consists of the two java classes `ScribeApplicationFactory` and `ScribeApplication`. The first class ensures that every participating node of the network runs an instance of the Scribe application. In the second class, the methods `join` and `leave`, which enable the node to join and leave the Scribe application in the overlay network, are defined. Also, all basic methods, which realize the application specific behavior (`createGroup`, `joinGroup`, `leaveGroup`, `sendMulticastMsg`) are declared in `ScribeApplication` class. These methods in turn call specific corresponding operations that form the second functional unit of Scribe implementation.

These operations are

- `ScribeLeaveOperation`,
- `ScribeCreateGroupOperation`,
- `ScribeJoinGroupOperation`,
- `ScribeLeaveGroupOperation`
- `ScribeMulticastOperation`
- `ScribeJoinOperation`

Fig. 10 presents an UML diagram of this part of implementation. Basically, all operations call the methods of functional unit three with the help of the method `executeOne`. The third functional unit adapts the general application specific behavior to the Pastry overlay. The class `PastryHandler` contains all Pastry specific methods, which are called by the method `executeOne` of the according operation**:** `join`, `leave`, `createGroup`, `joinGroup`, `leaveGroup` and `sendMulticastMsg`. These methods execute the `route` method of Pastry, which contains the according key and Scribe message defined by the user of the application. The information, which key and Scribe message type to use are passed down from the user to the methods of the application layer. With these specifications in the application layer, the simulator is enabled to trigger the behavior of Scribe application according to incoming events specified by the user. In order to process and react on this behavior, the Pastry implementation needed to be adapted on the overlay layer as well.

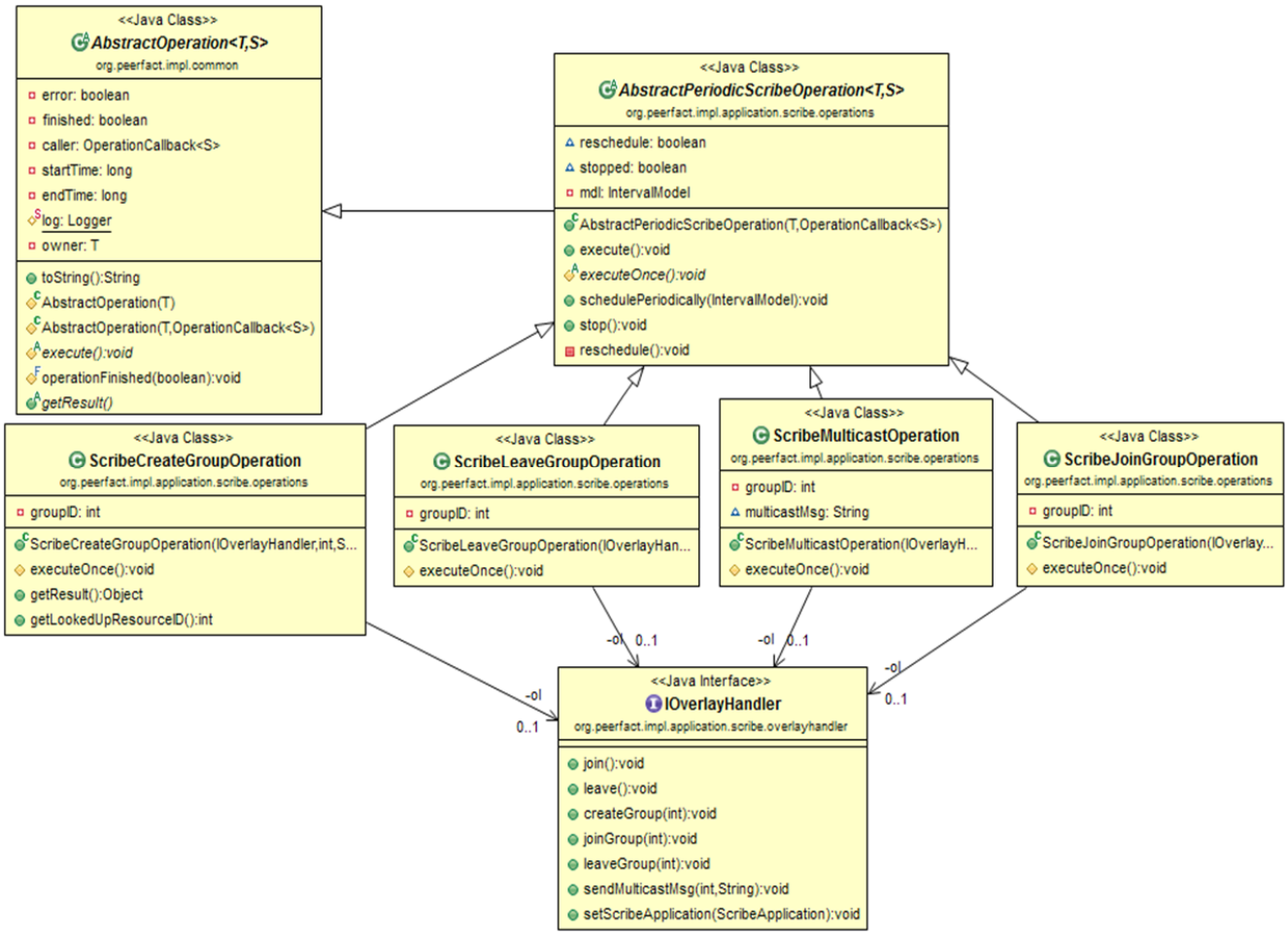

Figure 10. UML diagram for operation classes of Scribe´s implementation in application layer

2) Adaption of the Overlay Layer: The implementation of Pastry on the overlay layer contains several functional units, of which only two needs to be adapted.

In the first functional unit, the four kinds of Scribe messages, which use the method `route`, need to be defined at the overlay layer implementation of Pastry. The definitions of the messages are realized in the following java classes `SCRIBEcreateGroupMsg`, `SCRIBEjoinMsg`, `SCRIBEleaveMsg`, and `SCRIBEmulticastMsg`. The second functional unit defines the basic components and functionalities of the Pastry algorithm. The Processing of the Scribe messages are defined in the class `PastryMessageHandler`, which handles incoming messages and dispatches or processes them in every Pastry node. The `forward` and `deliver` methods of the original Scribe specification are also implemented there for every Scribe Messages. Fig. 11 presents the UML diagram of a part of Scribe implementation in overlay network.

Furthermore, the class `PastryNode` has been adapted in order to maintain the message groups and children. Hence, the class `PastryMessageHandler` reacts on incoming Scribe messages and calls group management methods of `PastryNode`. Thus, the Pastry implementation of the simulator has been enabled to process and react on incoming Scribe messages.

3) Simulation Scenario: In order to adapt the simulator to the newly implemented application, a configuration file had to be created. In configuration files, every component of the simulator can be defined. For the desired Scribe application, an entry of the Scribe application had to be added to the application component of the simulator, as well as an entry of the Pastry overlay to the overlay component. Furthermore, basic parameters of a simulation scenario can be defined, such as total number of nodes or overall simulation time. Since our implementation aims to create collaborative group up to 20 members in a short-term scenario, the variable size, which presents the number of the nodes, has been set to value 20. Fig. 12 illustrates the variables of the Scribe configuration file.

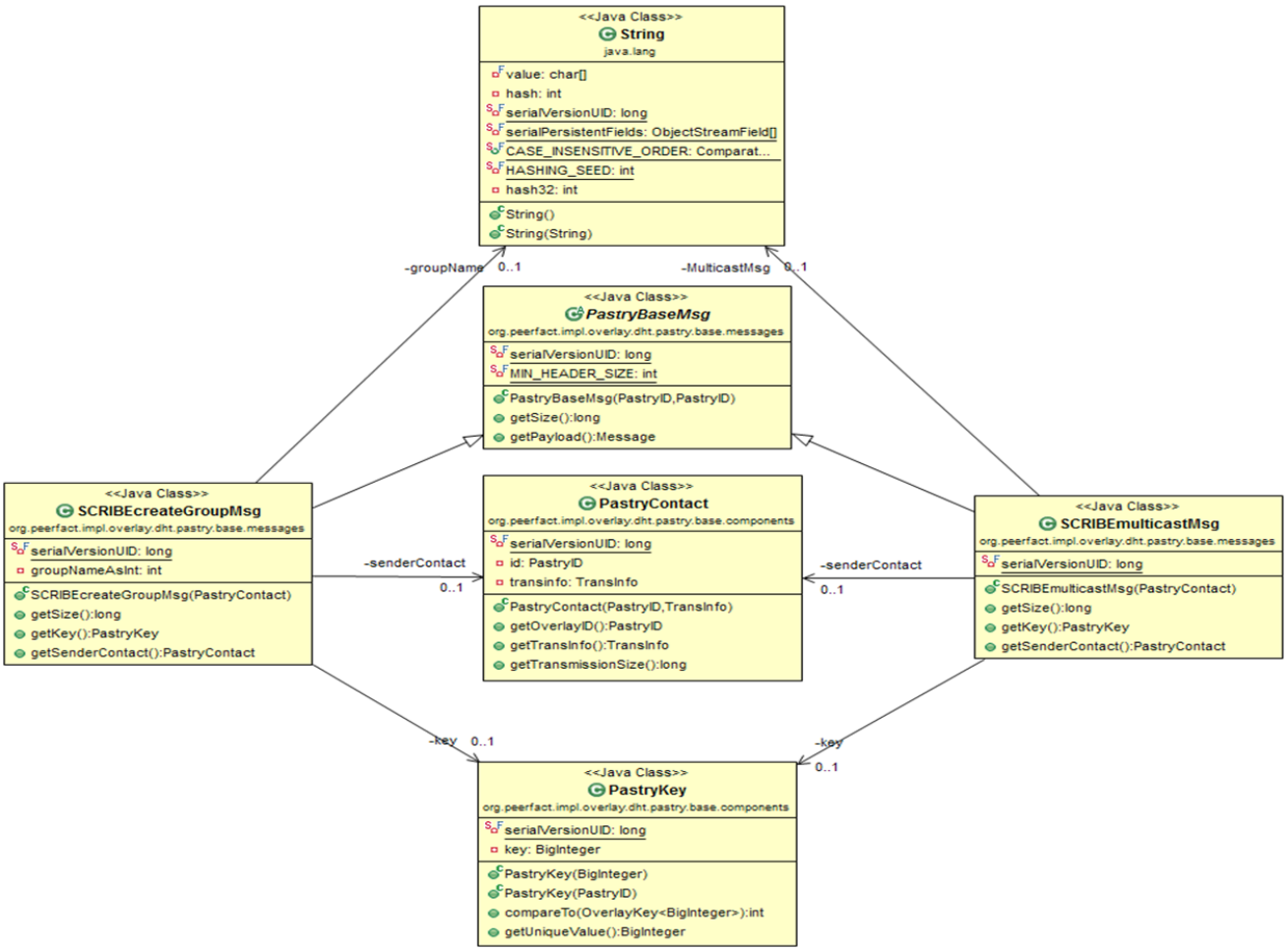

Figure 11. UML diagram for SCRIBEcreateGroupMsg and SCRIBEmulticastMsg classes at overlay layer

Every simulation scenario comprises several events, which need to be simulated. In the case of Scribe these could be: a node creates a group, another node joins the group, a third node sends a multicast message to the group, etc. In order to trigger these events in the simulation, an action file must be created. In the action file, all events are described by according nodeId, simulation time, application tag, and the events themselves. The simulator then processes the action file and triggers the according behavior at the application-layer by directly calling the methods of the class `ScribeApplication`.

```
<?xml version='1.0' encoding='utf-8'?>
<Configuration  xmlns:xi="http://www.w3.org/2001/XInclude">

    <!-- Variables -->
    <Default>
        <Variable name="seed" value="0" />
        <Variable name="size" value="20" />
        <Variable name="startTime" value="0m" />
        <Variable name="finishTime" value="100m" />
        <Variable name="ChurnModel" value="" />
        <Variable name="IsolationModel" value="" />
        <Variable name="NetLayer" value="SIMPLE" />
        <Variable name="Overlay" value="Pastry" />
        <Variable name="server" value="false" />
        <Variable name="Application" value="SCRIBE" />
        <Variable name="GUI" value="true" />
    </Default>
```

Figure 12. The variables of the Scribe configuration file

To test the implementation of Scribe, which is still in progress and need to be analyzed with better monitoring tools by the simulator, the following simple scenario has been accomplished. In the Fig. 13, the action file, which contains the simulation scenario of Scribe application, can be shown. In the scenario, there is a node called *one* as well as nineteen nodes called *world* summarized to a group called *all*. In the first minute of the simulation, node *one* joins the Scribe application. Between the minutes 2 and 40, the nineteen remaining nodes join the Scribe application. At minute 45, node *one* creates a group with groupID 200dec, which is randomly generated. Between the minutes 50 and 80, all remaining nodes join the created group. Finally, the node *one* sends a multicast message to the created group in the minute 90.

```
# action file for SCRIBE




one 1m ScribeApplication:join
all 2m-40m ScribeApplication:join

one 45m ScribeApplication:createGroup 200

all 50m-80m ScribeApplication:joinGroup 200

one 90m ScribeApplication:sendMulticastMsg 200
```

Figure 13. Scribe Simulation Scenario

To be able to investigate the functions of the new implemented Scribe application, PeerfactSim.KOM simulator provides the possibility to collect the output data and write them into a file that can be used for plotting with the tool `Gnuplot`. The Fig. 14 presents a line graph showing the number of initiated operations in the simulation scenario according to the time of simulation.

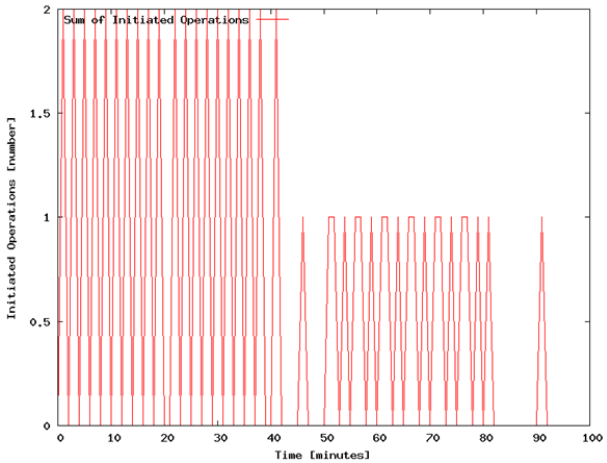

Figure 14. The number of initiated operations of Scribe application in the time of simulation scenario

After all nodes joined the Scribe application at the minute 40, one operation is executed in the minute 45, which is supposed to be the operation of creating a multicasting group. The operations of joining the multicasting group by the host group *all*, which consists of nineteen nodes, have taken place between the minutes 50 and 80 as it is explained in the action file. Whereas, the last prong at the minute 90 embodied the sending of multicast message.

On the other hand, the Fig. 15 shows the number of messages sent by the peers themselves to maintain the overlay and to join the Scribe application during the time of the simulation scenario.

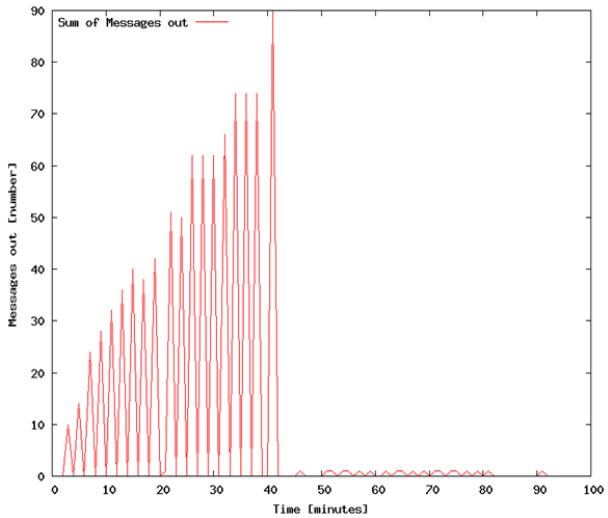

Figure 15. The number of sent messages in the time of simulation scenario

As mentioned in the action file of Scribe, all nodes have to be joined the Scribe application before the minute 40. That means the messages traffic during this time will be more active than after minute 40, which is illustrated in the Fig. 15.

The first results of this implementation showed some needs for more monitoring and analyzing functions of Peerfactsim.KOM simulator like the existence of a visualization analyzer and multicast message analyzer. Furthermore, the implementation has to be improved by extending the multicast message and testing it with many created groups.

VI. Conclusion and Future Work

In this paper, an overview of the current P2P collaborative environments and their use case has been presented. The P2P technologies have been briefly highlighted. We have also introduced our PeCoCC framework to allow computer-supported collaborative learning based on pure P2P networks that provide fully distributed and server-independent P2P communications and services, which increase the availability of these services and solve the problem of single point of failure present in server-based systems. The PeCoCC framework is currently in implementation phase. It supports interoperability and is being implemented in Java using the integrated development environment eclipse. To evaluate the performance of the framework, a P2P simulator named PeerfactSim.KOM is used. This simulator has a similar layered architecture like the PeCoCC framework and supports many forms of messages to communicate among the layers in the host. It is also implemented in Java and it offers a user-friendly graphical user interface. This simulator lacks some important protocols and components that will be needed to evaluate the PeCoCC framework. One of these protocols is the Scribe overlay multicasting protocol, which is also highlighted in this paper. The main components of the Scribe multicasting protocol have been implemented in the PeerfactSim.KOM simulator environment.

In the future, more extensive simulations of Scribe will be performed, i.e., simulations with a large amount of created groups, adding churn to the network, extending the multicast message to be able to contain more complicated content and investigating the network load and message routing latency by some collaborative scenarios. Furthermore, the structure of the PeCoCC framework, i.e., management and control module, overlay selection module and the PeCoCC collaborative messaging scheme have to be refined.

References

[1] M. Hasan and J. Seitz, "Peer-to-peer communication for computer-supported collaborative learning. The PeCoCC framework," In the Sixth International Conference on Mobile, Hybrid, and On-line Learning (eLmL2014) IARIA, Mar. 2014, pp. 25-29, ISSN: 2308-4367, ISBN: 978-1-61208-328-5

[2] G. Woodill, "Computer supported collaborative learning in education and training: Tools and Technologies," Phil. Trans. Brandon Hall Research. San Jose. CA. USA, 2008.

[3] M. Hauswirth, I. Podnar, and S. Decker, "On P2P collaboration infrastructures," Proc. 14th IEEE International Workshop on Enabling Technologies: Infrastructure for collaborative Enterprise, IEEE Press, June 2005, pp. 66-71, DOI: 10.1109/WETICE.2005.47.

[4] J. Vassileva, "Harnessing P2P power in the classroom," Proc. 7th International Conference, ITS, Aug. 2004, pp. 305-314, doi:10.1007/978-3-540-30139-4_29.

[5] M. Miettinen and J. Kurhila, "EDUCOSM – Personalized writable web for learning communities," Proc. Information Technology: Coding and Computing, ITCC, Apr. 2003, pp. 37-42, DOI:10.1109/ITCC.2003.1197496.

[6] C. Qu and W. Nejdl, "Interacting the edutella/JXTA peer-to-peer network with web services," Proc. IEEE Symp. Application and the Internet, 2004, pp. 67-73, doi:10.1109/SAIT.2004.1266100.

[7] T. Smith, "Sharepoint 2013 user´s guide: learning microsoft´s business collaboration platform," 4th ed, Apress, 2013, ISBN: 978-1-4302-4833-0.

[8] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," Communications surveys & tutorials, IEEE, vol. 7, 2005, pp. 72-93, DOI:10.1109/COMST.2005.1610546.

[9] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location and routing for larg-scale peer-to-peer systems," In 18th IFIP/ACM International Conference on Distributed Systems Platforms, Nov. 2001, pp. 329-350, ISBN: 3-540-42800-3.

[10] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structure peer-to-peer overlay networks," Proc. 5th Symposium on Operating Systems Design and Implementation, ACM Sigops Operation Systems Review, 2002, pp. 299-314, DOI: 10.1145/844128.844156.

[11] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," IEEE Journal on Selected Areas in communications, vol. 20, pp. 100-110, Oct. 2002, DOI: 10.1109/JSAC.2002.803069.

[12] J. Nogueira, "A large-scale and decentralised application-level multicast infrastructure,". [Online]. Available from: http://www.gsd.inesc-id.pt/~ler/docencia/tm0607/slides/Scribe-JoaoNogueira.pdf, 2014.12.01.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shanker, "A scalable content-addressable network," Proc. conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM 01, 2001, pp. 161-172, DOI: 10.1145/383059.383072.

[14] S. Ertel, "Unstructured P2P networks by example: Gnutella0.4, Gnutella0.6,". [Online]. Available from: http://ra.crema.unimi.it/turing/materiale/admin/corsi/sistemi/lezioni/m3/m3_u2_def/ceravolo_file2.pdf. 2014.12.01

[15] D. Boukhelef and H. Kitagawa, "Efficient load balancing techniques for self-organizing content addressable networks," Journal of Networks, vol. 5, Mar. 2010, pp. 321-334, doi:10.4304/jnw.5.3.321-33

[16] M.F.M. Firdhous, "Multicasting over overlay networks – a critical review," International Journal of Advanced Computer Science and Applications, vol. 2, pp. 54-61, Mar. 2011, ISSN 2156-5570.

[17] S. Ratnasamy, M. Handley, R.M. Karp, and S. Shenker, "Application-level multicast using content-addressable network," In the third International COST264 Workshop on Networked Group Communication, Oct. 2001, pp. 14-29, ISBN:3-540-42824-0

[18] "The FreePastry tutorial,". [Online]. Available from: https://trac.freepastry.org/wiki/FreePastryTutoriaal, 2014.12.01

[19] A. Montresor and M. Jelasity, "PeerSim: a scalable P2P simulator," Proc. IEEE Ninth International Conference on Peer-to-Peer Computing, 2009, pp. 99-100, DOI: 10.1109/P2P.2009.5284506.

[20] "PeerfactSim.KOM documentation,". [Online]. Available from: https://sites.google.com/site/peerfactsimkom/documentation, 2014.12.01.