Implementation of UMSIC Group Management Service

Jussi Laakkonen, Tommi Kallonen, Kari Heikkinen, Jari Porras
Lappeenranta University of Technology
P.O. Box 20,
53851 Lappeenranta, Finland
{jussi.laakkonen, tommi.kallonen, kari.heikkinen, jari.porras} @lut.fi

Abstract

Mobile devices make it possible for different users to communicate and collaborate with others despite their location. As the devices may have a connection to other devices but are not necessarily connected to the Internet, they should not rely on centralized services, but they should be able to handle the tasks in peer-to-peer manner. One of the important tasks with mobile device collaboration is group management. The groups need to be created and made available for other users to join as well as managed during and after the collaboration. In this paper we introduce the UMSIC project and the JamMo application designed for children's musical collaboration to enhance social inclusion. JamMo application relies on UMSIC middleware on many of it's features, especially the networking functionalities which are implemented in service oriented manner. The services provide the required functionalities for finding other users, to create groups with them and to create music within these groups. The group management service is in charge of creating, advertising, joining to and leaving from a group. Group management service as well as other UMSIC middleware services are designed to function in Peer-to-Peer manner without the need of centralized control.

Index Terms: UMSIC, Middleware, Service, PeerHood, Peer-to-Peer, Group Management

I. Introduction

The UMSIC (Usability of Music for Social Inclusion of Children) [1] project is a FP7 project funded by European Union aiming to develop an interactive product to enable children to communicate informally with their peers by using familiar mobile technologies. The purpose of the project is to improve social inclusion and reduce isolation in groups of children, especially targeting children with attention deficiencies and children of immigrants, whose language is different from that of the host country. The goals of this project are going to be achieved through musical activities in different social contexts [2]. These contexts include working alone, with pair and in groups either in school environment or on free time to create music with the tools offered. The product of the project, the JamMo (Jamming Mobile) application, is developed to answer the requirements [3] with the help of UMSIC middleware [4], [5]. The JamMo application relies on top of the middleware utilizing functions provided, which include automated connection establishment in service oriented manner (with the help of PeerHood [6]), secured data transmissions between devices, management of groups and manipulation of the music material. JamMo application will be used as stand alone application as well as in collaboration with other users. In collaborative use cases there is a need for group management. Collaborative work with JamMo should be possible anywhere when two or more JamMo devices are in same neighborhood. Therefore, the group management can't rely on centralized services, but all devices must be able to manage groups in a Peer-to-Peer manner. JamMo is designed and developed for Nokia N900 Internet tablet [7] running Linux based Maemo OS (Operating System) and licensed under GPL (Gnu General Public License).

PeerHood is an implementation of Peer-to-Peer neighborhood and communications concept in mobile environment which enables proactive discovery of devices and their services from neighborhood whilst providing means for communication. PeerHood is targeted to Personal Trusted Devices (PTD) and it is designed to be a transparent networking module between the network layer and the applications. The main components of PeerHood are the background daemon and the application library. PeerHood supports variety of networking technologies through modular plug-in structure. Currently WLAN (Wireless Local Area Network), Bluetooth and GPRS (General Packet Radio System) plug-ins are implemented. PeerHood offers functionalities for device and service discovery along with service advertisement, seamless connectivity between services and monitoring of nearby devices. The relation of PeerHood to UMSIC project and JamMo application services is presented in [8]. The benefits of PeerHood to UMSIC project and JamMo application are presented in [9].

Group management in wireless and mobile environments has been studied from different approaches. Ren and Boukerche [10] proposed a distributed group management scheme for ad hoc networks based on trust levels. AGAPE middleware [11] is a context-aware solution for group management in mobile ad hoc networks. Distributed group management solutions have also been widely used with wireless sensor networks. For example Vieira and Rosa [12] presented a reconfigurable group management middleware service for such usage.

In this paper the current state of implementation of UMSIC middleware and especially the group management service is presented. The solutions are presented from service implementation point of view, since the connectivity in UMSIC middleware is handled in service oriented manner utilizing Peer-to-Peer connections [5], [8].

II. JAMMO APPLICATION

One of the results of the UMSIC project is JamMo application for music creation and sharing. JamMo allows children of different age groups (3-6 and 7-12) to create and share music using different methods in four different usage scenarios: stand-alone, ad hoc, public and networked. In these scenarios, depending of the nature of the scenario, children can sing, compose with predefined sound loops or play with virtual instruments. For group and pair works the material is located on all participating devices as prerecorded sound data. If some user has recorded a audio sample to be used in group work it will be transferred to other participants before the work starts.

A. Stand-alone: Singing and composition games of 3-6-year-old children

The singing game is a simple karaoke application where a child can select a song and the sing along. The singing will be recorded and it can be listened later on. In the composition game child begins by selecting a theme from available possibilities. Each theme has a different look and different musical elements available. The composition happens by adding prerecorded sound loops, designed to fit to the backing track, to a musical track. There are two tracks; a backing track, which can´t be edited and a track where the loops can be added. On the user interface the loops are presented by different symbols fitting to the theme. A child can add, remove or move the sound loops to different parts of the track. After the song is finished it can be listened as whole and it will be saved to song bank on the device. The song can also be sent to teacher if the composition happened in a classroom environment.

B. Ad hoc: Composition pair game of 3-6 year-old children

The pair game is a extension to previously presented stand-alone composition game for 3-6-year-old children. The main difference here is that now the players are working as a pair to create a song. Before the composition is started child has to select a available group to join or he/she can create a own group for others to join. During the composition they now have three tracks in a song, one backing track, which cannot be edited, and one track for each player. Both can add sound loops only to their own tracks, edit own track and listen to a single track or tracks or the whole song independently. Information about all of their track edits (loop identifiers and places of the loops on the track) are transmitted in real-time to their pair, so both can see and listen what the other is doing. After the composition has finished, the song is saved on both devices and can also be sent to teacher.

C. Public: Inclusive music classroom of 7-12 year-old children

In public scenarios the use of JamMo happens in a classroom where a teacher has a computer with desktop version of JamMo and a video projector to present JamMo usage on a public screen. The children can work in groups up to four persons to create songs. They can create more complex songs than younger children by editing up to six tracks. There is one track for each participant and two tracks are reserved for teaching purposes, e.g. one has a backing drum track and the other contains basic melody for the song. They can add sound loops or sound created with virtual instruments to editable tracks. Additionally children can use own samples they've recorded with the devices, if the samples are not present they will be transferred before the work starts. The changes are updated to other members of the group in real-time. The teacher can monitor and control the activities of different groups from his/her computer.

D. Networked: Informal on-line community of 10-12-year old children

The networked scenarios are meant for older children and for non-real-time collaboration. Children can share musical material through on-line community, where children can also create workshops. The work basically similar to the group composition for 7-12 year-old children, the difference is that updates don´t happen in real-time, but they happen through server when the user logs in on the device. This way the work is informal without the need for teacher participation or control.

III. Connectivity in UMSIC Middleware

In the three latter scenarios (B, C and D) presented in previous chapter, the users need to communicate with each other. To handle this communication GEMS (GEneric Middleware Services) module of UMSIC middleware was created. GEMS handles all communication, profile and group related tasks through specialized services.

A. Single connection approach

In the UMSIC middleware a single connection between peers is utilized. Our implementation of Peer-to-Peer neighborhood and communications concept, PeerHood [6] is built on principle that it creates a new connection for each service it tries to connect to on remote device. As presented in [8] the middleware consists of multiple different services on each device. Since the middleware is built for mobile device (Nokia N900 Internet tablet) where memory usage should be carefully designed as recommended in Maemo developers manual [13], it would be a waste of resources to create a new connection every time some of the services is used. In addition PeerHood (Linux implementation written in C++) creates a new connection object for each established connection. If there are multiple devices in the neighborhood which have JamMo running the vast amount of connections could slow down the application and cause unnecessary memory fragmentation, which is discouraged in [13]. The memory fragmentation would occur when devices are in the move or there is interference in the wireless network and devices continuously connect (create object, allocate memory) and disconnect (delete object, free memory) to all services found on remote device. In JamMo more processing power must be reserved for handling audio material and not for middleware operations.

In order to utilize only a single connection we have created a procedure where the established connection object is passed to other services on lower levels, the hierarchical order of services was presented in [8]. The connection established by PeerHood is a two-way TCP socket connection and therefore, it doesn't matter which of the devices has initialized the connection. For example when device A finds a device B which has service JamMo running and device A creates a connection to that particular service there is no need for device B to create a connection to the JamMo service running on device A. The connection created by device A to service on device B can be utilized for further communication. The PDU (Protocol Data Unit) of sent packet contains information about the destination service so messages to different services over single connection can be distinguished and dispatched to proper handler.

B. Connection establishment

First the device searches for other devices from the neighborhood which have JamMo service enabled. The main purpose of the JamMo service is to verify the software versions running on both devices, at this point user doesn't have to be logged in yet. If there is no pending connection (waiting for software verification or peer authentication) or existing connection to that particular device a new connection will be established to the service on the newly found device.

After the versions of the software are verified through JamMo service the created connection object is passed to Authentication and Authorization service which verifies the users on the both devices and exchanges certificates of both users. After authentication a symmetric AES (Advanced Encryption Standard) encryption key for the session between the two connected devices is exchanged. In the current implementation of the UMSIC middleware the Authentication and Authorization service is not yet implemented in this extent. After the authentication procedure the connection is finally fully established and can be used for further communication between the two devices. The connection object is moved to list of connected devices on both devices.

When the connection is in established state and can be found from the list of connected devices the middleware tries to request the user profile (the profile information contains public personal information about the user on that device) from that device. A request is sent to Profile service on remote device. The Profile service is a simple service following request-reply approach, which handles incoming requests and replies with requested information if the requesting user has proper authentication. If the user hasn't logged in on the device to where the request was sent, nothing will be returned as there is no active profile. To make sure that user profiles are always available the profile requests are performed on regular intervals to each connected device if profile is not received.

IV. Group management in UMSIC Middleware

After the users are identified, authenticated and profiles are exchanged, it is possible to establish groups of different sizes between users to create music together. This is done by using the group management service of the UMSIC middleware. After the group is established and the game is started the actions of the users are transferred between the group members using collaboration service.

A. Group management between peers with centralized controller

For establishing groups between peers Group management service has been implemented. It is designed to support distribution of group information to peers and to allow users to join to a group and leave from a group. When a user creates a group and becomes the owner of the group (created by the middleware when the user decides to start a networked game) the advertisement of this particular group is started. This advertisement is sent to all connected devices (to Group management service) on regular interval and stopped when the group is locked (creator of the game selects to lock the game and after this further joins to the group are not possible) or when the game is stopped and group deleted. The receiver processes the group advertisement and stores (or updates if such group already exists) the information into list of groups so the user can see what kinds of groups are in the neighborhood. The middleware also performs periodic requests to connected devices to inquire about their current group status. When Group management service receives this kind of inquiry it responds with current group information message which contains same information as the advertisement sent by group creator. The answer to an inquiry can be sent by anyone who is in some group, user does not have to be the owner of that particular group. When a response to inquiry is received by the Group management service it will be added to the same list with group advertisements or existing group info is updated if such group exists. The group information is removed from the list of groups if no advertisement or response to inquiry is received after defined period of time, currently after one minute. This is done to keep the size of the list fairly small and to remove the groups which have been closed, deleted or moved away from range.

The group works as a Peer-to-Peer group, each member of the group has a connection to each group member. This removes the need for centralized control of group actions (e.g. adding a loop to a track), each member of the group sends all group actions directly to other members of the group. But we do have a loose ownership to each group for maintaining the group memberships in a centralized manner. In current solution the ownership can be transferred to some other member of the group and therefore, labeled as loose ownership. If the owner of the group decides to leave the group, ownership is given to next member in the group (next is the next one in the group member list) and the group work can continue as before. Other members of the group receive an advertisement with the new owner defined in the message data and they update their group information with the received information. These kind of changes are allowed only from the current owner, which in this case is the leaving owner. Also new member notifications or leaves, which force members of the group to update their current group information, can be originated only from the current owner of the group.

B. Benefits of the centralized controlling in group management

The centralized controlling of group memberships reduces the amount of messages sent at joining phase. For each join the additional messages needed to be sent is the amount of users before the newly joined user as shown in Figure 1. If the group was controlled by peers it would require that each joining member sends join message e.g. only to one member of the group that was defined in the previous group advert or inquiry response. After this it would require that existing members of the group negotiate about the joining of the new user. This procedure is presented in Figure 2, where all devices notice that the advert is coming from Device 1 and therefore, are sending the requests to that particular device.

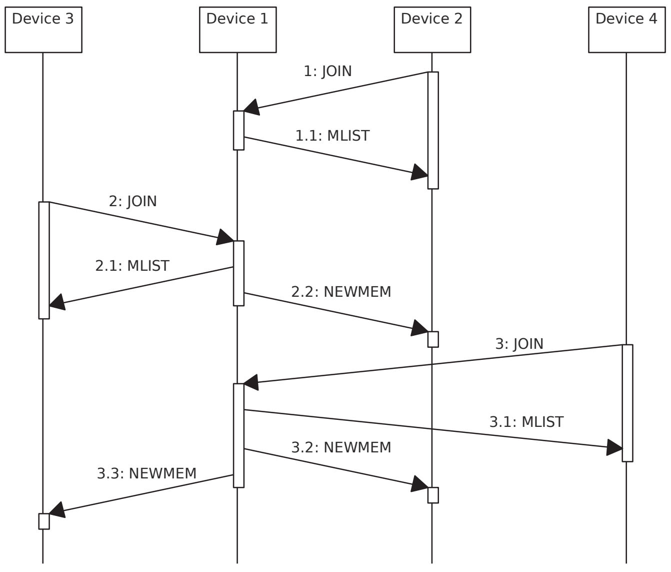

Figure 1. Join procedure when group is controlled by Device 1

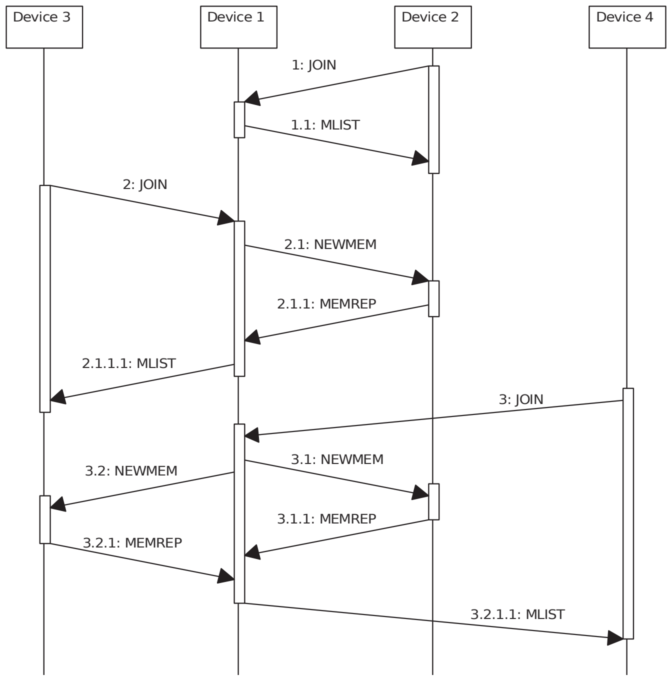

Figure 2. Join procedure when group is controlled by peers

From Figure 2 it can be seen that each join in a group controlled by peers would require two messages between joiner and connected member (request and reply) and the negotiation messages between each member. This would result in

$$2 + 2(n - 1) \qquad (1)$$

messages where *n* is the amount of members in group at that moment. The centralized controlling of group is lighter from network consumption point of view because it requires

$$2 + (n - 1) \qquad (2)$$

messages for each join or leave (as it can be seen from Figure 1), where *n* is the amount of members in group before join.

Since the main usage of JamMo application is in classrooms there can be multiple small groups (pairs and/or groups of 3 or 4 as specified in [2]) the amount of group management messages should be kept as small as possible because devices are connected through single WLAN [4]. The target device, Nokia N900 Internet tablet [7], has also a Bluetooth adapter but it is not used in the project because of the bandwidth limitations [4]. As the processing of the audio material is resource consuming in general and especially on a mobile device, the transferring of the different actions will require a lot from the network in terms of low latency, especially when real time actions are required (less than 50 ms is recommended by Zimmerman et. al. [14] for tolerable musical performance and less than 10 ms is required for the needs of the UMSIC project [15]).

In figures 1 and 2 message JOIN (join request) is 12 bytes, NEWMEM (new member notification) and MEMREP (reply to new member notification) are 16 bytes each and size for MLIST (member list, in bytes) can be calculated with

$$12 + 4(n - 1) \qquad (3)$$

where *n* is the group size at that moment. The sizes and contents of the messages are defined in the protocol specification of the UMSIC project [16]. The amount of data to be sent at each joining phase can be calculated by using equations (1) and (2) and setting correct sizes for messages. For peer controlled groups it will result in

$$12 + (12 + 4(n-1)) + ((16 + 16)(n-1)) \qquad (4)$$

where n is the amount of group members in the group at that moment. In centralized controlling the amount of data can be calculated with

$$12 + (12 + 4(n-1)) + (16(n-1)) \qquad (5)$$

where n is the amount of group members in the group at that moment. Figure 3 shows the cumulative amount of data sent when a user joins to a group. In Figure 4 the effect on current group sizes is shown.

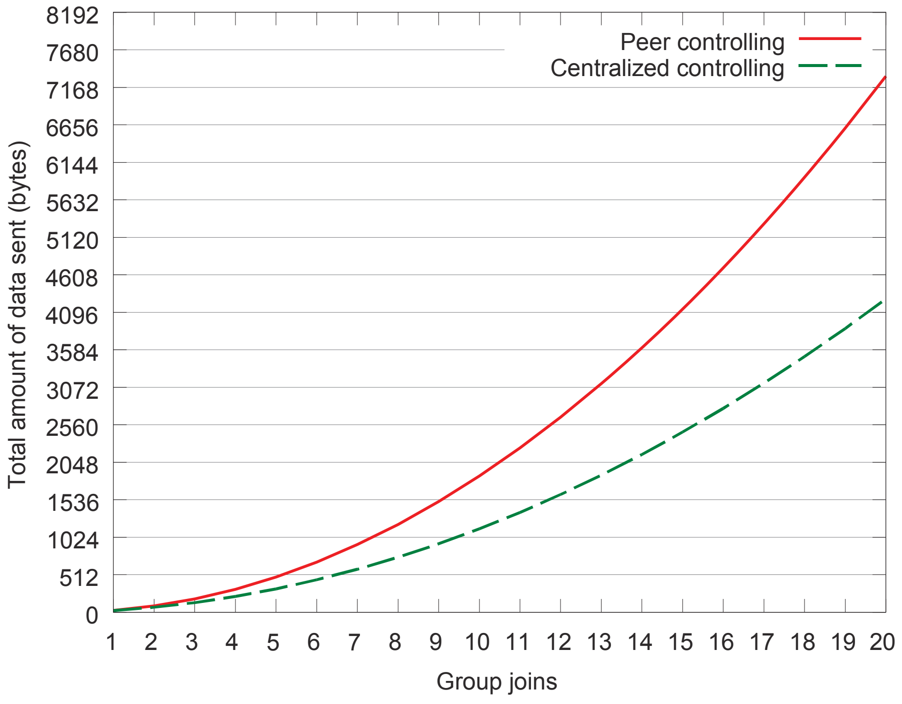

Figure 3. Total amount of group management data sent after each join

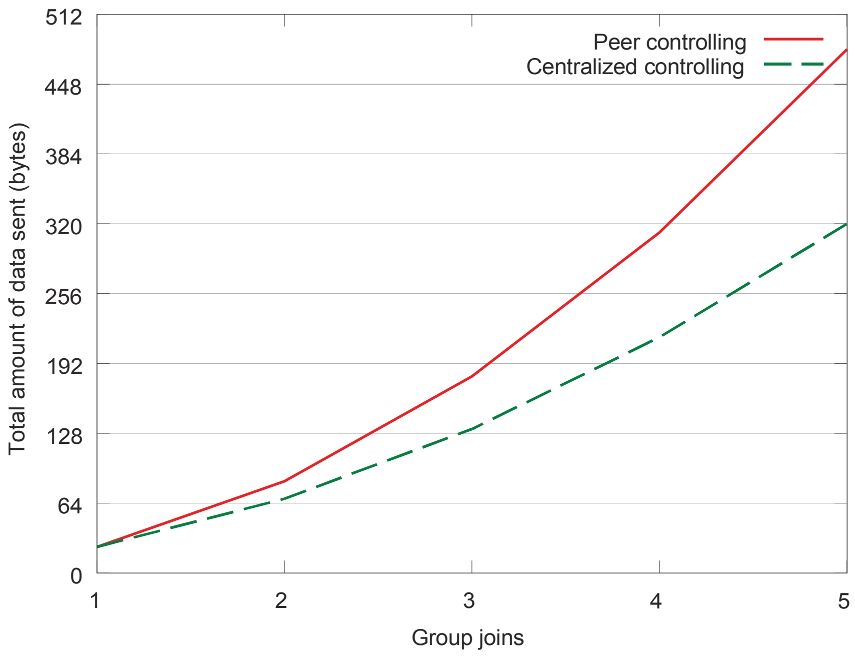

Figure 4. Total amount of group management data sent after 1st, 2nd, 3rd, 4th and 5th join

From both figures (3 and 4) it can be seen that the difference is fairly minimal when group sizes are small (less than 8). As the size of the group grows the difference is quite significant when compared to the total amounts of data to be sent between group members. The effect of our selected approach to the network consumption is therefore, quite minimal, since in the scenarios presented in chapter II the maximum group size is 4.

In the classroom the tasks given for different students can vary in length of the task and in task difficulty. The amount of group management messages can grow rapidly since new group is established for each task. When the amount of connecting devices grows (e.g. in a classroom there could be over 20 devices communicating in small groups) the amount of group management messages naturally grows depending on the sizes of the groups.

C. Collaboration during group work

After the group has been successfully created and other users have joined it the responsibility of managing the the group work moves to collaboration service. The collaboration service is only active during group work — if the group work is not active, collaboration service discards all messages sent to it. During active group work the collaboration service is used to transfer the actions of a user to all other group members. If a user does a change in his/her track (each user has only one personal track he/she can edit), information about the change is sent to other group members. The change can be adding or removing a loop on loop track, or a change on virtual instrument track. The information about the music on virtual instrument tracks can be either midi information about note on/off at certain time or in the case of slider instrument the used frequency at certain time. There is no need to transfer actual audio data between devices (except singing, which is transferred as a whole when it is finished), for our needs it is enough to transfer the control data related to changes, since all audio material is the same on each device. These changes do not need to be transferred in real time, but rather "as-fast-as possible" to keep the application usable.

Each user action generates certain amount of data, which is sent to every group member directly. A loop action (add, remove and move) generates 22 bytes of traffic [16] that will be sent to each peer in group. When virtual instruments are played, the actions are sent as midi data where each note generates 29 bytes of traffic [16] to each peer in group. If a slider instrument is played each action requires that a 32 byte sequence [16] is sent to other group members. Since the nature of the tasks varies a lot and the behavior of the users is hard to predict, it is not feasible to try to estimate the real amount of traffic to be sent between group members. The slider instrument generates most traffic, since it is played by moving a finger on touch screen [15] and each change is sent to other group members (32 bytes each) [16]. These changes can happen multiple times per second (from 20 to 100) depending on the behavior of the user. With the slider instrument the amount of data created by one user during one second can reach maximum of 3200 bytes. The amount of data to be sent by all members of the group at once can be calculated with

$$b\,(\,n-1\,)\,m \qquad (6)$$

where b is the amount of data to send, n is the group size and m is the amount of users playing an instrument at the same time. If there are four members in one group and each of them uses a slider instrument the group alone could generate maximum of 38,4 kilobytes of traffic each second.

After the users have finished the song, the song is saved locally on all devices and collaboration service is disabled. The information about the users taking part in song creation is saved with the musical data. This way the information about group is not lost even though the group as such doesn't exist anymore.

V. Conclusion and future work

The presented approach for connectivity in UMSIC Middleware suits well for the purpose since memory usage is put to minimum. Based on early tests basic networking operations do not seem to be slowing down the application. Solution for group management in UMSIC Middleware works for the needs of networked games as it was seen on early tests. As it can be seen from equations (1) and (2) the difference of amount of messages grows linearly and therefore, full benefit of our approach could be seen with larger groups. This applies also for total amount of group management data sent as Figure 3 depicts. The full effect on overall performance is yet to be measured since the project is still going onward and development of the application continues. The measurements are conducted in the future along the stress testing of WLAN networks with multiple JamMo applications simulating playing of slider instrument in different groups. The stress tests will be performed to see how many devices can work together in the same network with real-time requirements (latency stays below preferred limits). The results of the future tests will give more detailed information about the bottlenecks of our protocol and implementation.

Acknowledgment

The authors would like to thank the UMSIC project.

References

[1] M. Fredrikson, J. Porras, J. C. Read, P. Paananen, S. S. Elmer and G. Welch. “The UMSIC project homepage”, [online], www.umsic.org.

[2] The UMSIC Project, “Work Package 1 - Requirements for social situations”, *University of Jyväskylä, Finland*, Deliverable 1.2, 2009.

[3] The UMSIC Project, “Work Package 1 – Software requirements for JamMo”, *University of Jyväskylä, Finland*, Deliverable 1.5, 2009.

[4] The UMSIC Project, “Work Package 2 – Requirements specification of the UMSIC middleware”, *Lappeenranta University of Technology, Finland*, Deliverable 2.1, 2009.

[5] J. Laakkonen, T. Kallonen, J. Porras and K. Heikkinen, “System and Architecture requirements for UMSIC middleware”, *Wireless World Research Forum 23rd Meeting*, Beijing, China, 2009.

[6] J. Porras, Petri Hiirsalmi and A. Valtaoja, “Peer-to-Peer Communication Approach for a Mobile Environment”, *Proceedings of the 37th Hawaii International Conference on System Sciences*, Hawaii, USA, 2004.

[7] Nokia, “Nokia N900 mobile computer – Technical specifications”, [online], http://maemo.nokia.com/n900/specifications.

[8] J. Laakkonen, T. Kallonen, K. Heikkinen and J. Porras, “Introducing UMSIC Middleware Services”, *Proceedings of 6th Seminar of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program,* Helsinki, Finland, pp. 53–60, 2009.

[9] J. Laakkonen, T. Kallonen, K. Heikkinen and J. Porras, “Usability of Music for Social Inclusion of Children, System and Architecture Requirements for UMSIC Middleware”, *IEEE Vehicular Technology Magazine*, vol. 5, no. 1, pp. 55-61, 2010.

[10] Ren, Yonglin, Boukerche, Azzedine; "A secure group management scheme for mobile ad hoc networks," 2010 IEEE Symposium on Computers and Communications (ISCC), pp.429-432, 22-25 June 2010

[11] D. Bottazzi, and A. Corradi, Eds., "Context-Awareness for Impromptu Collaboration in MANETs", Proc. Of WONS, pp. 16-25,2005.

[12] Vieira, M. S. and Rosa, N. S. 2005. A reconfigurable group management middleware service for wireless sensor networks. In Proceedings of the 3rd international Workshop on Middleware For Pervasive and Ad-Hoc Computing (Grenoble, France, November 28 - December 02, 2005).

[13] Maemo.org, "Maemo Diablo Reference Manual for Maemo 4.1" [online], 2009, http://maemo.org/maemo_release_documentation/maemo4.1.x/Maemo_Diablo_Reference_Manual_for_maemo_4.1.pdf

[14] Zimmermann, R., Chew, E., Ay, S. A. & Pawar M. "Distributed musical performances: Architecture and stream management", *Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*. Volume 4 Issue 2. 2008.

[15] The UMSIC Project, "Work Package 1 - Requirements for music making", *University of Jyväskylä, Finland*, Deliverable 1.1, 2009.

[16] The UMSIC Project, "Work Package 6 – Report of JamMo Extension Development", *University of Oulu, Finland*, Deliverable 6.4, 2010.