A Redundancy Information Protocol for P2P Networks in Ubiquitous Computing Environments: Design and Implementation

Rafael Dias Araújo, Hiran Nonato Macedo Ferreira, Pedro Frosi Rosa, Renan Gonçalves Cattelan
Universidade Federal de Uberlândia
Uberlândia, MG, Brazil
{*rdaraujox,hirannonato*}*@gmail.com*, {*frosi,renan*}*@facom.ufu.br*

***Abstract*—The ubiquitous computing vision brings high computational and communication demands. In this paper, we propose a high availability protocol for information replication in ubiquitous computing environments. Running at the application layer, on top of a P2P overlay network based on JXTA, our protocol allows the transfer of multimedia contents automatically generated by multimedia capture systems. We formalize the characteristics of the protocol and present its design rules and procedures.**

***Keywords*-Redundancy protocol; P2P networks; ubiquitous computing; multimedia content sharing.**

I. INTRODUCTION

The concept of ubiquitous computing [15] has been widely used nowadays. In particular, automated systems that allow the capture of live experiences are a recurring research theme [1]. Classrooms instrumented with electronic whiteboards, microphones and video cameras produce multimedia artifacts that reconstruct the captured experience for future use and review. In order to reach true ubiquity, the produced contents should be available to users in a transparent way, i.e., independent of their physical location.

In instrumented settings for multimedia capture, we face the problem of high computational and communication demands [8]. Traditional implementations reported in the literature have scalability problems caused by centralized entities that become the system bottlenecks [3]. Satyanarayanan [10] notes that issues like remote access, high availability, power management, mobile information access have increased and, in parallel, ubiquitous and pervasive computing take advantage of distributed and mobile computing. Thereby, systems must provide high availability [13] to ensure the transparency requirements. The main aspect of high availability is the redundancy of information that must be transmitted to different points of the network by reliable communication channels.

To solve these issues, we developed a peer-to-peer (P2P) architecture for the storage and distribution of captured multimedia content. P2P networks have the potential to make the process of sharing information much easier. Studies show P2P are responsible for more than 50% of the overall Internet traffic in some regions [12]. Another key advantage of P2P networks is the direct availability of resources to the network participants, without the need for any central coordination by servers or stable hosts [11]. Furthermore, its robustness is increased because it removes the single failure point commonly observed in a client-server based solution [7].

In order to contribute to the proper capture and storage of multimedia content, it becomes indispensable the creation of a protocol that ensures communication and interoperability features for heterogeneous devices. Thus, in this paper, we propose a P2P based protocol that aims at minimizing the aforementioned problems of large data transfers and ensures the availability of the information captured from the environment. We based our approach in the general idea of cooperation among user devices joining a P2P network.

The remaining of the paper is structured as follows: in Section II, we present a real multimedia capture scenario which inspired the design of our protocol; in Section III, we detail the abstraction layer defined for the protocol; in Section IV, we describe our protocol specification and design, i.e., its environment, encoding, vocabulary, services and procedure rules; in Section V, we present the implementation details behind our approach; in Section VI, we present related works; and finally, in Section VII, we make our final remarks.

II. CLASSROOM INFORMATION CAPTURE

Consider classrooms equipped with electronic devices (e.g., mobile phones, notebooks, tablets, electronic whiteboards, video cameras, etc.) and responsible for capturing multimedia raw data from the environment. The resulting captured data, in the form of multimedia artifacts, may be useful both for instructors that need to reuse them later and for students to review what was presented.

Take, for example, iClass [9], an open-source capture platform for ubiquitous learning environments. iClass comprises a federation of capture clients and an access servers. Capture clients are software components for a particular capture device and generates a corresponding multimedia artifact (e.g., an audio capture component monitors a microphone and generates audio streams). Access servers are daemon applications which collects multimedia artifacts sent by capture clients and merges then into a single, synchronized

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-183-0

document. Users have access to those documents by using an integrated Webserver. iClass infrastructure is thus predominant based on a client-server approach: clients produce content which is sent to a server for user access.

iClass presents characteristics that provide some interesting insights for our research context:

- The captured data have widely varying formats such as video, audio, image and text.
- Data reaches large volumes over time, thus requiring scalable and high availability software and hardware infrastructures.
- Moreover, it is important that storage do not be centralized and be made in a reliable way to protect users' personal annotations.
- Finally, it is also worth to observe that, once created and stored, such data artifacts usually do not change. This characteristic allows a simpler data replication policy and reduces consistency problems.

With these characteristics in mind, we extended iClass original architecture by adapting it to the P2P paradigm. We conceived the concept of capture agents, software components with well defined interfaces that capture information and distribute it not through a single, a priori known server, but through an access service run by cooperative peers on a P2P overlay.

III. Content Abstraction Layer

In order to abstract the content transferring services, we created a layer using the widely available JXTA open-source P2P protocol [14]. The main goals of JXTA are: operating system independence, language independence and providing services and infrastructure for P2P applications.

We use JXTA services to make content transferring transparent to the application. Such approach creates an abstraction capable of aggregating and providing many services for content storage and synchronization. The proposed layer was named CAL (Content Abstraction Layer).

Thus, capture peers implement the JXTA protocol to communicate among themselves and intermediate peers may be used to route messages to external peers without direct connection. For this, there is a concept that should be explored: peer grouping. In JXTA, a peer group is defined as a collection of peers that have agreed upon a common set of services. Each peer group is identified by a unique peer group ID and each one can determine its own membership policy from open to highly secure and protected (credentials are required to join). Peers may belong to more than one peer group simultaneously.

When devices finish a capture session, they can search for an available storage service running on some peer group by using the *discovery* service. Devices need to join the network only when they wish to transfer content, i.e., they can work offline while capturing content.

IV. Protocol Specification and Design

The five essential protocol elements (environment, vocabulary, encoding, services and procedure rules) [4] are described in this session. The environment takes into account the physical and logical characteristics where the protocol is used, as architecture, computer's organization, etc. The vocabulary is the set of events (name of messages) used to specify the state transitions of protocol and the encoding the format of each message. The service is an abstract element that defines a feature and its behavior is defined by procedure rules (automaton).

CAL was designed to be used over P2P networks and each peer can perform both roles of producer or consumer. The producer is the one that captures the real world data through user devices. The consumer is the one that stores the data received from the producer and also shares them with other consumers. Thus, peers can share their contents to provide redundancy and high availability on the network.

Another aspect of this layer that should be considered is the connection-oriented networking, i.e., the upper layer must first establish a communication session with the other node and, after that, it becomes capable to deliver data in the same order it was sent [4].

After these considerations about the environment, we formalize the protocol vocabulary, which defines the semantic of messages used in communication [4]. Our protocol's set of messages consists of:

- LIST_STATUS_REQUEST: message sent to a peer to request a list of its contents. This message can be sent only by consumer peers. This is the first step for two peers to synchronize content between themselves;
- SEND_SEG_REQUEST: message used to send a content segment. When the content to be sent is to large, then they have to be broken into smaller pieces before being sent;
- SEND_SEG_RESPONSE-: this message represents a SEND_SEG_REQUEST unacknowledgement. It is sent when the last SEND_SEG_REQUEST was not recognized. Note that the positive SEND_SEG_RESPONSE is not adopted because this message is used by a service with negative acknowledgment [4];
- FT_GET_REQUEST: message used to request a specific content using its identifier obtained by calling the *list_status* service. This message can be sent only by the consumer;
- FT_PUT_REQUEST: message used to send a content that was just created. This message can be sent only by the producer;
- SEND_MSG_RESPONSE+: this message represents an acknowledgment. It is used to indicate that the last SEND_MSG_RESPONSE primitive was properly recognized. This message represents the positive response for the JXTA SEND_MSG service;

- SEND_MSG_RESPONSE-: this message represents an unacknowledgement. It is used to indicate that the last SEND_MSG_RESPONSE primitive was not properly recognized.

These messages are encoded by a character-oriented method and have the same format and comprising header, body and trailer.

In the message header, there is a flag named *MORE_BIT* that is used to indicate whether there are more segments to be transferred. This situation can occur when a consumer wants to synchronize to another and the second has so many files that it exceeds the maximum primitive's length, or when a producer wants to send a content so large that it need to be broken into smaller pieces. The *SERVICE* field represents the called service's name that is encoded in a binary number. As CAL has seven available services, then three bits are enough for this field in order to represent all services ($2^3 = 8$). Finally, there is a field to store the length of the current content and, hence, the data field can be variable. The trailer stores the data hash code for further error checking.

Now, we can define the services provided by CAL and present its behavior. As previously mentioned, the layer contains eight services: *publish*, *start_session*, *end_session*, *list_status*, *ft_get*, *ft_put*, *send_seg* and *reject*.

The first one, *publish*, is used to make a new consumer peer available on the network. It means the new peer wishes to provide its disk space to store content for other peers (acting as a server). The peer uses this service to become available for connection to other peers. This is not a confirmed service, i.e., the peer sends this primitive and do not expect a confirmation.

The *start_session* service represents the connection establishment phase from a peer to another. Regardless of the role played by the peer in the network, this phase is mandatory. Fig. 1 shows the automaton for this service. The application that uses CAL requires a connection by calling the service mentioned above. This service starts the connection establishment process. The first action is to find some available peer in the network. For this, CAL uses the *discovery* service from JXTA platform and waits the *discovery_confirmation* with the needed peer's information to allow the communication. If timeout is reached, it retries *n* times, where *n* is a previously defined constant. When the service receives a positive *discovery_confirmation*, it calls the *connect* service from JXTA and waits a confirmation, which can be positive or negative. In the first case, i.e., if it receives a positive *connect_confirmation*, it goes to the *CONNECTED* state. Otherwise, or if timeout, it tries to discover and to connect to other peer during the pre-established number of attempts.

As the previous one, the *end_session* is also mandatory and represents the disconnection phase. When the application wants to close the connection with the other peer, it calls this service and, then, CAL instantiates the *disconnect*

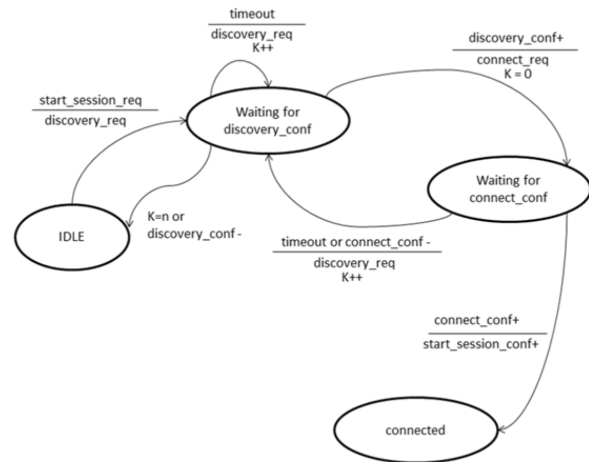

Figure 1. *start_session* service behavior

service from JTXA, returns to IDLE state and become ready to establish a new connection to another peer. This behavior is shown in Fig. 2.

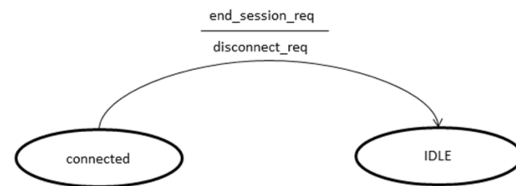

Figure 2. *end_session* service behavior

The *list_status* service is available only to the consumer peer. This service is responsible for asking the other connected peer what contents it has stored. As a result, the application obtains a list of contents identifiers. Once connected, the application sends a *ls_request* (*ls* is an abbreviation for *list_status*) and CAL sends a *send_msg_request* with the *ls_request* inside its data field. CAL waits for the confirmation that can be positive or negative. If the incoming message is a negative confirmation, it returns to *CONNECTED* state. If the incoming message is a positive confirmation, CAL sends a *send_msg_response+* to confirm the message. If the received message has any error, the *send_msg_response-* is sent to refuse the message. This confirmation message is sent inside the *send_msg_request* data field. After this message, CAL becomes ready to receive the list. There are cases that the peer has so many files that the list must be partitioned in segments to be sent as small pieces, as shown in Fig. 3. In this case, each segment is sent with the *MORE_BIT* flag equals to 1 until there is no more segments to be sent. This automaton also considers the timeout while waiting for the *ls_confirmation* message, while waiting for a segment or while receiving a segment.

For clarity, Fig. 4 shows the temporal order diagram for

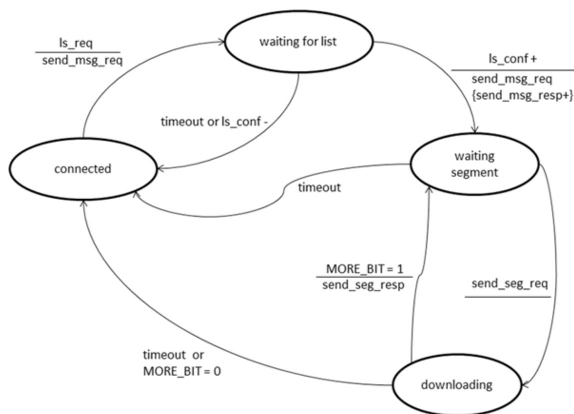

Figure 3. *list_status* service behavior

the *list_status* service. One can observe that the incoming messages at CAL are transmitted inside the data field of the *send_msg* primitive of JXTA. This behavior can also be observed in other services, once they were designed to have grater abstraction from those provided by JXTA.

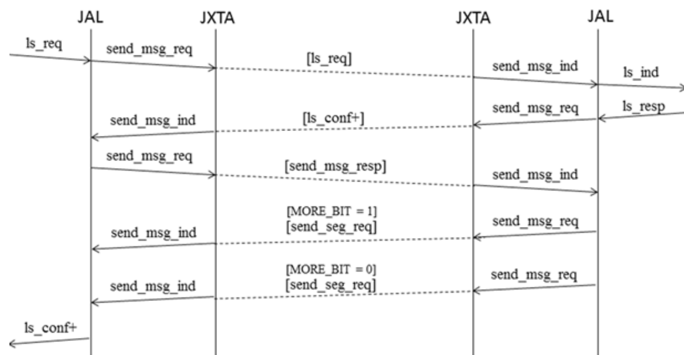

Figure 4. Temporal order diagram for *list_status*

The *ft_get* service is also only available to the consumer peer. It is used to request a specific content transfer, using the identifiers that were obtained by calling the *list_status* service. Fig. 5 shows the *ft_get* service behavior. When the application is already connected, it can request the transfer of some content by sending the *ft_get_request* primitive and, then, CAL sends a *ft_get_req* within the *send_msg_request* primitive data field. If it receives a negative *ft_get_confirmation*, it returns to the *CONNECTED* state, otherwise it sends a *send_msg_response+* message to confirm the receipt of the primitive and indicates that it is ready to receive the segments of the required content. Thus, the segments are sent until the whole content is transferred. Note the flag *MORE_BIT* to indicate that there are more segments to be sent. When the transfer is finished, CAL sends a confirmation message to the application indicating the content transferring is complete.

The *ft_put* service is available only to the producer peer.

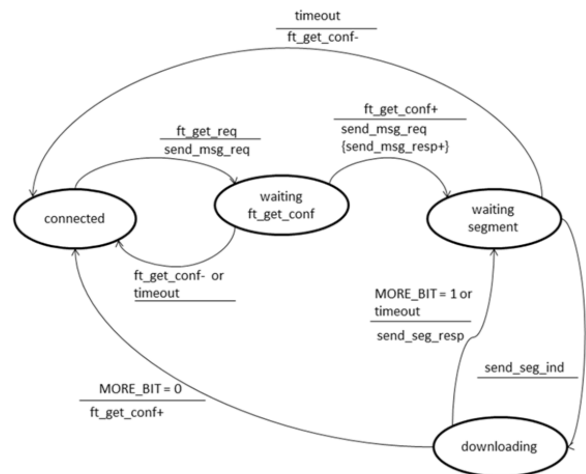

Figure 5. *ft_get* service behavior

This service is used to send contents to be stored in other peers (those who are consumers). When the application is already connected, it sends a *ft_put_request* message to CAL and waits confirmation from the consumer. If CAL receives a negative confirmation, it returns to the *CONNECTED* state and indicates to the application that the transfer has failed. This happens to let the decision of trying to resend the content to the same peer or connect to another peer as a responsibility of the upper layer. If the received message is a positive confirmation, it becomes ready to receive the content segments. Once finished the segments transferring (i.e. *MORE_BIT* = 0), CAL goes to the *CONNECTED* state and sends a positive *ft_put_confirmation* message to the application, indicating that the content upload was done. This service automaton is shown in Fig. 6.

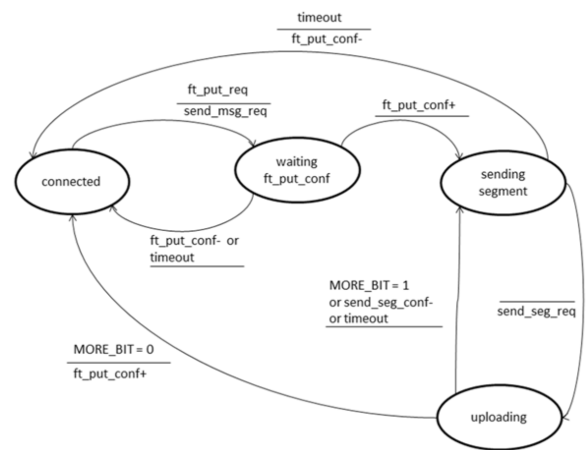

Figure 6. *ft_put* service behavior

The *send_seg* is available to both producer and consumer peers. It is responsible for sending segments of contents. In this scenario of multimedia content sharing, the data are

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-183-0

usually too large to be sent in a single transfer. So, data must be broken and sent in small segments, as indicated by the *MORE_BIT* flag. Its behavior is shown inside the automata of the previously presented *list_status*, *ft_get* and *ft_put* services.

An interesting behavior observed in *list_status*, *ft_get* and *ft_put* services refers to the message confirmation technique used while sending the content segments. We opted for a negative acknowledgment approach, which means that CAL do not wait for a positive confirmation for each *send_seg_request*. Instead, CAL receives a *send_seg_response-* when an error is found in the transmitted message.

Finally, the last service named *reject* is also available to both producer and consumer peers. CAL can receive faulty primitives or worse, it may receive some primitive whose service should not be recognized by the peer in question, i.e., those messages that are only recognized by peers who play a particular role. In these cases, this service must be used to refuse the mistaken incoming primitives. Eventually, these errors may cause side effects in CAL because there could be burst errors that the link layer is not able to handle in consequence of false positives. The service behavior is very simple. In every state of all services mentioned before, the layer may receive mistaken primitives from the application. Then, if it happens, CAL sends a message with the *reject* service and goes back to the same state and waits another message. Therefore, it ensures the consistency of communication between the application and CAL.

V. Implementation Details

Our initial implementation was written in Java. One of the main reasons for this choice is due to its portability and wide use by the community. In this section, we describe the prototype implementation of our P2P protocol and discuss several implementation decisions that were taken during its and design and development.

As previously mentioned, CAL sits between the JXTA platform and the P2P application. Our implementation was proposed this way in order to give full autonomy for the P2P application to create its own content transfer policies. Thus, the transferring information rules are not defined by CAL but by the P2P application.

Each service in CAL (*publish*, *start_session*, *end_session*, *list_status*, *ft_get*, *ft_put*, *send_seg* and *reject*) was implemented through a well-defined interface, allowing applications to use its functionality with the least possible effort. The implementation details of each service are shown below.

publish: This interface is invoked to provide a storage service. It uses native JXTA primitives to publish an advertisement. Advertisements are XML documents that describe network resources [14]. Our publish interface consists of only one method named `publish()` that receives the provided service's name.

start_session: This interface is used to establish a connection to a storage service available on the network. Among the various available JXTA services, this interface uses `discovery` and `connect`. One of the greatest potential abstractions of our protocol is achieved through this service, because the network search happens completely transparently to the application. It is optional to the peer querying the network to inform or not the storage service it wants to connect to. If the connection is successfully established, a JxtaBiDiPipe is created between the two peers to perform the various connections about to come. The JxtaBiDiPipe uses the core JXTA uni-directional pipes (InputPipe and OutputPipe) to simulate bi-directional pipes in the J2SE binding [14].

end_session: This interface is responsible for finalizing a connection with a storage service. The `end_session()` method is responsible for disconnecting the pipe created in `start_session()` and releasing the peer to create new connections or simply cease to exist.

list_status: The consumer peer uses this interface to know the contents of a peer storage service. The communication between the peers happens through the messaging service provided by JXTA, which is implemented by the `StringMessageElement()` method. The main methods used for this service are: `getMessageElement()`, `addMessageElement()` and `sendMessage()`. In this interface the data are trafficked through content segments. A *MORE_BIT* flag is used to indicate if there are more threads for each request. After all segments are received, the information is collated and reported to the application as previously requested.

ft_get: This interface is called by a consumer peer that wishes some content that is unavailable in its repository. Content is sent through the network in segments, thus employing the same methods used by the `list_status` service. Among the various methods used by this interface, we can mention the `check_free_space()`, which is used to check whether there is enough free disk space to receive such file.

ft_put: This interface is used by producers to send content to a storage service. As in *ls_status* and *ft_get*, contents are sent into segments in the network, using always the *MORE_BIT* flag to identify the last segment.

send_seg: This interface is responsible for segmenting the content and sends it over the network. As previously mentioned, it is used by various services of this protocol, as: `list_status`, `ft_get` and `ft_put`.

reject: This interface is called whenever a new packet is received. it is used to inform the sender that the received package has some error and then it was rejected. This interface, as well as some mentioned above, also uses the messaging services provided by JXTA to send it inside its data field.

VI. Related Work

The use of P2P platforms in ubiquitous computing systems is not a completely new idea. For example, eComP [6] focuses on designing P2P networks for everyday objects. eComP is a decentralized XML-based messaging system that abstracts the underlying network and communication protocol and provides services through well-defined interfaces. Similar to our JXTA-based approach, the underlying network infrastructure requires no fixed infrastructure or the support of any other entity except computing peers. However, a user-defined ID (like a familiar, personal or rational textual name) is deemed necessary to identify resources.

Hong et al. [5] developed an effective scheme to manage multimedia sharing based on specially designed profiles and a virtual community. Their multimedia sharing layer is responsible for sharing multimedia contents, and is constructed as a specially designed scheme based on locality. They focuses on an effective community construction scheme performed by community construction layer.

Barolli and Xhafa present JXTA-Overlay [2], a JXTA-based P2P middleware for distributed and collaborative systems. JXTA-Overlay allows the integration of end devices, such as sensors and personal/mobile computers, providing transparency and security for sharing, contributing and controlling available resources. JXTA-Overlay comprise a set of primitive operations: peer discovery, resource allocation, file/data sharing, discovery and transmission, instant communication, among other services. Such primitive operations can be exchanged between connected peers and support different types of applications related to collaborative activities. Besides presenting a similar layered structure, differently from JXTA-Overlay, where the application must know the identity of peers to which it desires to connect, our protocol does not require any *a priori* knowledgement of peer identities. In our proposal, CAL provides such network abstraction and still ensures the reliability and security properties offered by JXTA during content transfers.

VII. Conclusion and Future Work

We presented CAL, a P2P-based protocol designed to transfer multimedia information captured by different types of devices installed in instrumented environments. As a result, devices that produce multimedia contents do not need to have any previous knowledgement of the network to be able to transfer contents to other devices. CAL creates abstractions for discovering peers offering storage capabilities with safety and reliability properties.

Complementing our approach and as future work, we are currently developing an effective access mechanism for retrieval of the multimedia information stored in our platform. Its query approach is based on contextual preferences that uses information available about users, devices and the environment in order to automatically recommend and personalize returned content.

Acknowledgments

The authors would like to thank the Brazilian Research Agencies CNPq and FAPEMIG for supporting this work.

References

[1] G. D. Abowd and E. D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7:29–58, 2000.

[2] L. Barolli and F. Xhafa. Jxta-overlay: A p2p platform for distributed, collaborative, and ubiquitous computing. *IEEE Trans. on Industrial Electronics*, 58(6):2163–2172, 2011.

[3] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proc. of the Intl. Conf. on Information Technology: Coding and Computing*, pages 205–213, 2005.

[4] G. J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[5] C.-P. Hong, E.-H. Lee, and S.-D. Kim. An efficient scheme to construct virtual community for multimedia content sharing based on profile in a ubiquitous computing environment. In *Proc. of the Intl. Joint Conf. on INC, IMS and IDC*, pages 1271–1276, 2009.

[6] A. Kameas, I. Mavrommati, D. Ringas, and P. Wason. ecomp: An architecture that supports p2p networking among ubiquitous computing devices. In *Proc. of the Intl. Conf. on Peer-to-Peer Computing*, pages 57–, 2002.

[7] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Commun. Surveys Tuts.*, 7(2):72–93, 2005.

[8] F. Perich, A. Joshi, T. Finin, and Y. Yesha. On data management in pervasive computing environments. *IEEE Trans. on Knowl. and Data Eng.*, 16:621–634, 2004.

[9] M. Pimentel, L. A. Baldochi Jr., and R. G. Cattelan. Prototyping applications to document human experiences. *IEEE Pervasive Computing*, 6:93–100, 2007.

[10] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Pers. Commun.*, 8(4):10 –17, 2001.

[11] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proc. of the Intl. Conf. on Peer-to-Peer Computing*, pages 101–102, 2001.

[12] H. Schulze and K. Mochalski. Internet study 2008/2009, 2009. http://www.ipoque.com/en/resources/internet-studies. Retrieved: Dec. 2011.

[13] Sun BluePrints Online. High availability fundamentals, 2000. http://www.sun.com.br/blueprints. Retrieved: Dec. 2011.

[14] Sun Microsystems, Inc. Project JXTA v2.5: Java programmer's guide, 2007.

[15] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.