Towards Flexible Data Sharing in Multi-Device Mobility Scenarios

Josef Spillner, Sven Bendel, Alexander Schill
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {josef.spillner,sven.bendel,alexander.schill}@tu-dresden.de

***Abstract*—For most people, computers have evolved from large stationary machines into a multitude of interconnected personal and mobile devices which can be and are used at any convenient time. Likewise, the handling of data has evolved from the traditional input-processing-output chain into more sophisticated flows which involve splitting, joining, duplication, migration and sharing. Users of multiple mobile devices hence require new techniques for being able to migrate and share data across device boundaries depending on the context. We present a desktop-level data communication technique for personal, social and collaborative mobile environments and discuss its implementation into existing devices and platforms.**

I. Motivation

Data sharing is an essential activity in personal computing. Data can be shared between applications on one device, between applications on multiple devices of one user, or even between applications on the devices of multiple users. Access to web applications and web services allows for even more sharing schemes. Sharing between local applications happens in memory by using a clipboard as copy/paste mediator with one active selection and multiple buffers, or by passing file references, for instance for temporary files created by downloads which are then picked up by the responsible application. Sharing between local and remote applications, as well as just between remote applications, is made possible by networked services which run on one of the devices or centrally in the network. Likewise, sharing between devices needs access to such a service, unless physical media is used to transfer the files.

Data sharing is distinguishable from file sharing and from application sharing. File sharing refers to replication of files for later use, whereas data sharing implies an instant use of the data for some purpose with or without a serialization to files. Application or document sharing assumes multiple users working on one synchronized document which is opened and potentially collaboratively edited by multiple networked applications in parallel. The notion of data in this paper is referring to both document contents and buffer contents. Fig. 1 visualizes the scope of data sharing.

The data handling in current personal environments such as desktops and mobile phone interfaces is insufficient and difficult for most real-time collaborative scenarios involving many devices and services. While files can be downloaded, then copied to another device and finally opened there manually, it would be desirable to be able to chain these actions automatically. An example for this requirement is a download of an electronic book on an Internet-connected mobile phone. The book should not reside on the phone, but rather open instantly on an eBook reader which has a better display, although it lacks Internet connectivity.

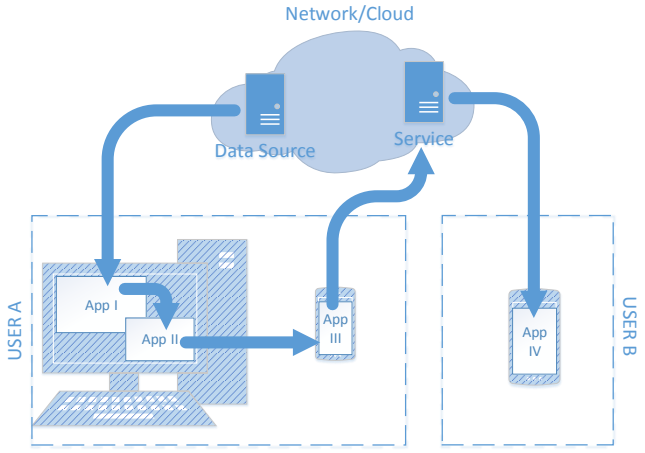

Fig. 1. Overview about the scope of data sharing with multiple devices, applications and services accessed over a network

A second difficulty exists due to devices, especially mobile ones, being in a volatile state of being switched on, online and reachable. Most actions assume a direct connectivity whereas in practice they fail when the connectivity is not present. Manual sharing processes typically involve removable media (for nearby users) and online sharing services (for both nearby and distant users). This convenience should be kept as much as possible, while the inconveniences, including a manual sign-up at a potentially untrusted online service, should be minimized.

A third difficulty is that social and trust relations are not taken into account. Neither are device capabilities. Hence, each cross-device action requires manual thinking about the risks and benefits of passing over the handling of a file to a particular device. Sometimes, a much more high-level process would be desirable. An example for this is 'share this photo album with all of my friends'. When a friend is reachable on

two devices, the most suitable one should receive the album. If a friend is not reachable at all, a web service should store the album for later retrieval instead.

A fourth difficulty results from the temporary nature of many data sharing situations. Often, a file is passed for glancing over it, or a picture is shown to somebody to provoke a quick comment. Currently, the receiving user is obligated to garbage-collect shared data because sharing mechanisms don't support timestamps or flags indicating temporal aspects. In social network research, timestamps have been suggested as a (rather ineffective) access control measure, whereas for data sharing they can serve as a useful hint for data decay.

In order to overcome these four difficulties, we propose OpenShare, a powerful delay-tolerant and decentral data sharing scheme which works across device-bound and online applications by taking the specific data, device and service properties into account. We claim and show that OpenShare can be implemented with today's tools, and we aim at demonstrating its usefulness through user testing in actual data sharing scenarios. In the next section, the current handling of data in various popular environments is described more precisely. A summary and rating of relevant findings from research on more flexible data sharing complements this analysis. Then, we present our intended concept for data handling between devices and services. We include a formalization of data sharing environments, an architecture description and a prototype discussion as first results of our work. Finally, the text is concluded with a future work plan which describes the evaluation.

II. Existing Data Handling and Sharing

In this background section, we explain media types, look at existing media type handling in desktops, followed by an anlysis of research on data sharing schemes and a summary of actual sharing systems.

MIME (Multipart Internet Message Extension) has historically been introduced to enable rich e-mail content beyond plain text. Over time, the handling of media types initially specified with MIME has become a core concept of desktop environments on PCs and mobile devices. The MIME media type registry of IANA, the Internet Assigned Numbers Authority, currently lists 1401 types in the type classes application (1014), audio (140), example (1), image (47), message (22), model (17), multipart (16), text (69) and video (75) (http://www.iana.org/assignments/media-types).

The file `mime.types`, introduced with the Netscape browsers, its `mime-support` database variant (789 entries) and the `shared-mime-info` package (http://freedesktop.org/wiki/Specifications/shared-mime-info-spec, 939 entries in 17 classes) are typical representations of MIME types for desktop applications. These media type databases map media types to file name patterns, content excerpts (so-called `magic` detection), aliases, preferred icons for their visualization and human-readable descriptions. A number of desktops and applications make use of `shared-mime-info`, including the open source desktop environments KDE, GNOME and XFCE, by complementing it with a database of installed applications and their data handling capabilities (so-called `mailcap`). File managers and web browsers are the dominant media type-dependent applications. They offer type-dependent context menus to let the user perform actions with chosen files, for instance 'open with (application)...', 'send to (device, user)...', generic local actions including 'compress'/'uncompress', 'encrypt', 'sign', 'show preview', 'submit to web' or 'sync with the cloud', and type-specific ones such as 'mount (ISO)' or 'extract pages (from PDF)'.

On other desktops, similar databases and actions are used. Apple Mac OS X offers the Launch Service which reads application and service `.plist` files and manages a central file associations database. Microsoft Windows uses registry entries which map to media types. On Google Android, there is a MIME media type database for web-related libraries and type-dependent intents which work like actions. The JavaBeans Activation Framework is intended to be used for encapsulating data access, data type determination and type-dependent operations in Java environments. Beyond desktops, media types are used in HTTP servers to specify the type of transmitted content.

In none of these desktops, type-dependent actions consider sharing workflows, device states and capabilities or trust relations.

Several researchers have proposed advanced data sharing schemes particularly for mobile devices. However, none of them ties into existing and widely used desktop and application concepts. XMIDDLE, a middleware to share and collaboratively edit XML documents with other users and reconcile them with global versions, has been one of the first support systems [1]. Chucking, a sharing technique to project screen contents from a mobile device onto a public display [2], comes closer to the intended use cases. However, Chucking inherently assumes a real-time interaction whereas our goal is to support passing data to devices which are currently switched off, or in private to users who are currently unavailable. Gradual Engagement [3] delivers a suitable design pattern but is not concerned with integration.

The synchronization of files depending on device capabilities has also been identified early on as a challenging topic [4]. Capability descriptions of services, in particular Software-as-a-Service (SaaS), exist in various formats, for example as instances of the Universal Service Description Language (USDL) or the Web Service Modelling Language (WSML). However, there isn't any suitable proposal yet on how to model application capabilities beyond media type handling. On the other hand, the capabilities of devices can be expressed with W3C's Composite Capability/Preference Profiles (CC/PP), Comprehensive Structured Context Profiles (CSCP) and further serialized models [5].

There are already some deployed alternatives for the sharing of files and other data. We will briefly analyze e-mail systems, cloud storage offerings and specific sharing services, and reason about why these systems are not adequate sharing

backbones.

Sharing over the network is often performed using e-mail out of convenience. The global distributed e-mail system has been tested, is in wide use and works reliable with client implementations for all end user platforms. E-mail also integrates nicely with web services to import and deliver data. Nevertheless, the e-mail system has not been designed for automated data sharing, and this shows in a number of undesired effects. Firstly, sharing tasks mix up with regular communication. This leads to tedious search or filter maintenance. Secondly, complex workflows need to be decomposed manually into individual message exchanges. Thirdly, devices cannot be distinguished by their capabilities. Fourthly, practical anti-spam measures cause unnecessary delays, malfunctions and a low quality of experience. These measures are often combined with inflexible quotas and strict capacity restrictions imposed by the hub providers.

Dropbox [6] being a popular representative for cloud storage at first sight allows seamless synchronization of files between different devices and platforms. They also provide an API for integration into other web services. But there are some major drawbacks when using Dropbox. Firstly, there is no way to decide on what to do with sent files on the destination device. Secondly, syncing must be manually initiated on the destination side because Dropbox does not automatically detect changes in the main repository on mobile devices unless per-application polling is used. This results in the need to wait for the device to sync which limits the perception of seamless sharing and increases power consumption. Thirdly, syncing objects other than files (such as pure text in buffers) is not possible. Fourthly, there is no selective sync to only certain devices. Folder sharing to other users is possible, while single file sharing is not. Fifthly, the user has to move a file into a special folder in the file system. To preserve precious online storage users also tend to remove the file after they are done with it.

Pushbullet [7] is a web service to push semantically distinguishable data such as notes, addresses, lists and files from a web browser, using a browser extension or a plain web frontend, to paired Android devices. It avoids interaction with the destination device and makes objects easy to find due to their appearance in the notification bar. However, it is limited to just Android platforms, excludes social structures to generate destination candidate groups, is not integrated into desktop environments and again requires a manual decomposition of workflows. Furthermore, it is a closed commercial application which makes it impossible to extend.

Hence, any of these deployed systems and technologies are insufficient for our goals. None of them supports workflows, web services and applications as sharing destinations, as well as advanced content-based sharing and interaction by recognizing the appropriate content types and device capabilities. Most require manual interaction with the destination device. Synchronizing cloud storage services in particular face major obstacles which can only be solved by smarter client-side sharing facilities.

III. Concept for Cross-Device Data Handling

We propose OpenShare, a feature-rich scheme to share data between two or more applications, devices or users. OpenShare is characterized by its delay-tolerant (queued) operation, by its usability in decentralized environments, and by its strong reliance on well-described data objects and device properties. Compared to existing mobile computing environments, it offers:

- Unified sharing. Objects resulting from buffers or files shall be handled equally to keep the learning curve short.
- Chained actions. Users can combine actions which are otherwise manually performed on each device. For instance, a composite action 'open on largest screen' selects the device with the largest screen, then copies the data to it and finally processes it with the default application.
- Device descriptions. Each device is thoroughly described in a machine-readable way. In particular, its non-functional properties such as trust, ownership and capabilities are considered. While device descriptions exist, they are so far not used for data sharing tasks, and need to be extended to capture sharing-specific properties.
- Application descriptions. These are intersected with device descriptions and matched against the media types of relevance in a sharing task. Thus far, no application description specification exists. In addition, native and web applications and web services should all be considered first-class citizens and treated equally in terms of user experience by using a more abstract definition of applications.
- Asynchronous actions. Users copy buffered data or a file to a device which is currently offline. As soon as the device reconnects to the network, it will receive the data.
- Storage service integration. Data which is queued is stored with high quality and security over a selection of appropriate services. In order to achieve the desired quality, dispersion and encryption techniques are applied.
- Access to the OpenShare functions as a service through a fault-tolerant proxy in the network, or through local publish-subscribe desktop services, e.g. D-Bus, COM or Distributed Notifications on the aforementioned desktops.
- Middleware support for asynchronous queue operations either between a centrally located storage service and the target devices, or in a peer-to-peer network in which each device operates a queue. We intend to use existing distributed social network protocols such as XMPP for this feature. Inter-application sharing, alias intelligent copy and paste between applications on the same device, will not need to access the central OpenShare services, and clients in the same subnet can perform peer-to-peer sharing without using a globally accessible OpenShare service.

The enclosing workflow of sharing activities is outlined in Fig. 2. The individual tasks can be conditionally executed. Most OpenShare features are well represented within the workflow, for instance the initial choice of whether an action

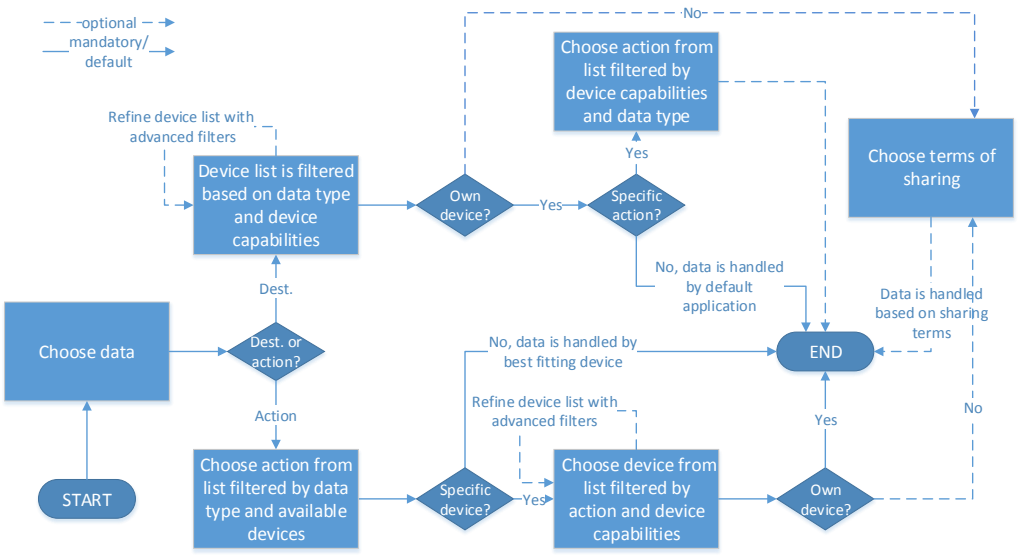

Fig. 2. Sharing workflow from a user's perspective based on fixed actions or destinations

or a destination is fixed.

Application and device capabilities are intersected to yield all possible and finally, through a ranking mechanism, the best possible combinations. This reduces the effort for the user to find the right device and application for the shared object. An intelligent sharing destination filter which works similar to the client-server matchmaking found in service-oriented brokers will be used to create this intersection. One challenge is that the filter will perform a three-way matchmaking not yet found in these brokers. It adds complexity by introducing three hard entity levels (user requirements, devices, applications) alongside soft destination selection via more filters – biggest display, closest mobile device and similar ones. The filters also need to derive not explicitly described attributes such as cost and performance differences when using a service instead of a local device as a target. This complexity poses problems on the design of the user interface and the algorithmic performance which we attempt to solve in the future, while this paper focuses on the coarse-grained sharing workflows.

IV. Formal Representation and Architecture

The realization of the OpenShare concept follows a formal software design technique. Initially, we suggest a suitable abstract and formal representation of the scenarios and goals we have in mind. Then, we derive a software architecture and explain its prototypical implementation.

A. Representation of cross-device sharing scenarios

We regard a set of users $U|u_1...u_n$ with mobile devices $M|m_1...m_n$ and online services $S|s_1...s_n$. All devices and some of the services offer personalized accounts M_U and S_U. All devices and accounts offer computational resources (storage, communication and computation) with different limitations. On each device and service endpoint, there are applications $A_M/A_S|a_1...1_n$ which operate on documents D with a certain document type T_D. All operations O are bound to the document type as O_T and take arguments which may include M, S and U. We do not consider device, application and data ownership in this model but acknowledge the need to secure all sharing transactions through appropriate means, including a permission management for sharing to devices of other users as well as web applications and services.

A user who wants to open a PDF file on a first device so that it displays on the second is modeled as follows: $m_{u1} : O_{T:=PDF}(D) = display(m_{u2}, a_2)$. A more complex workflow is the download of a video file from a public web application with subsequent time-limited sharing of an embedded key frame to another user: $m_{u1} : O_{T:=Video}(D) = download(S) \wedge O_{T:=Picture}(D') = store(m_{u2}1, duration)$.

B. OpenShare architecture

The architecture of the envisioned data sharing framework – Fig. 3 – consists of document and buffer operation handlers, access to a sharing middleware on each device and a device database to keep track of available devices and their capabilities. The database is populated by a device discovery tool but can be manually extended and corrected by a user in a one-time effort. We also assume an application database on each device. As outlined before, such databases already exist but need to be enhanced with specific capability attributes and extended to local web applications which are growing in popularity. The OpenShare middleware itself may be located centrally or distributed across all devices. One requirement is that it is permanently available from all devices which take part in a sharing task.

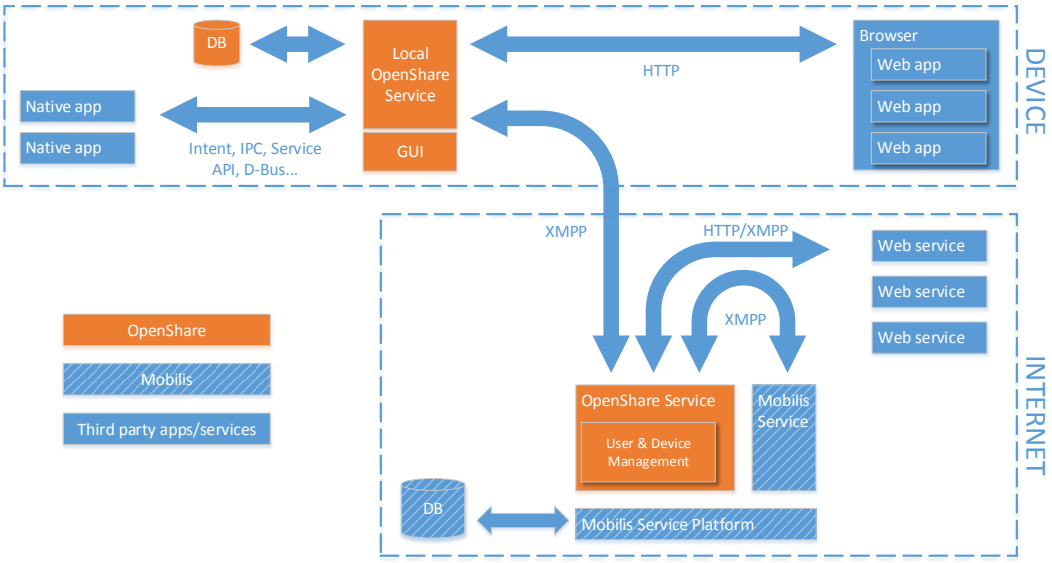

Fig. 3. OpenShare architecture with a central service and database accessible by all devices and decentral local services per device

C. Device database and discovery realisation

We have prototypically implemented OpenShare using a combination of existing prototypes and glue files for desktop integration.

A prerequisite for finding the right device is a mobile device database for which the use of a device discovery tool is essential. Several such tools exist which combine device searches over heterogeneous protocols, including Bluetooth, DNS-SD and WiFi. Webinos [8] and FlexiSource [9] are freely available for mobile device discovery. Further tools exist as implementations of Mobile Ad-hoc Networks (MANETs), for instance Ambient Talk [10], which cover more network functions including decentralized routing under changing environment conditions and device configurations. For OpenShare, we consider FlexiSource due to its support for device-bound capability and ownership descriptions. It is a device discovery framework written in Python with actual discovery plugins implemented in arbitrary programming languages. Each plugin scans for devices. Once found, it retrieves information about the computational resources and additionally, if possible, device-bound description files which express capabilities and ownership.

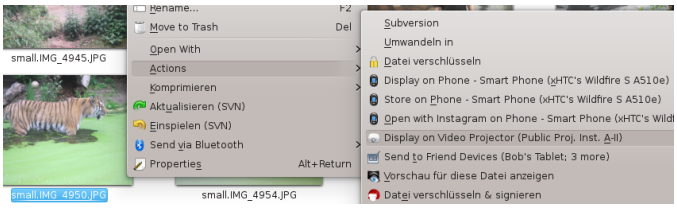

Fig. 4. Screenshot of the KDE desktop and file manager integration of OpenShare

All descriptions are then merged into one database per FlexiSource instance. For OpenShare, each active (sending) device hence either runs its own FlexiSource instance or accesses a central database in the network. The XML database contents are transformed into desktop-specific data handling extensions. For KDE, a file manager service menu and a clipboard action menu are created. Through them, an `openshare-client` is called with the target application, user or device as argument. Fig. 4 is a screenshot from the KDE file manager Dolphin whose context menus have been populated with actions from discovered devices.

D. Workflow realisation

Combining the convenient session and publish/subscribe characteristics with transfer capabilities for files and arbitrary data, the XMPP protocol suite in combination with occasional HTTP and local IPC calls delivers the lower-level connectivity layer of OpenShare. On top of XMPP, the Mobilis Platform [11] offers rich functionality for social and collaborative scenarios involving multiple users and devices. All devices can be dynamically maintained using FlexiSource. Through Fig. 3, the use of Mobilis for the central OpenShare service becomes obvious. Fig. 5 shows the previous workflow from the system perspective with more details on communication choices.

A first version of the OpenShare service has been implemented. The combination of this service, the Mobilis platform, the FlexiSource framework and the extensibility of contemporary desktops make more flexible data sharing possible. Apart from the features of OpenShare, we expect a practically usable prototype with several strong non-functional properties to promote its adoption. To promote its adoption, we will

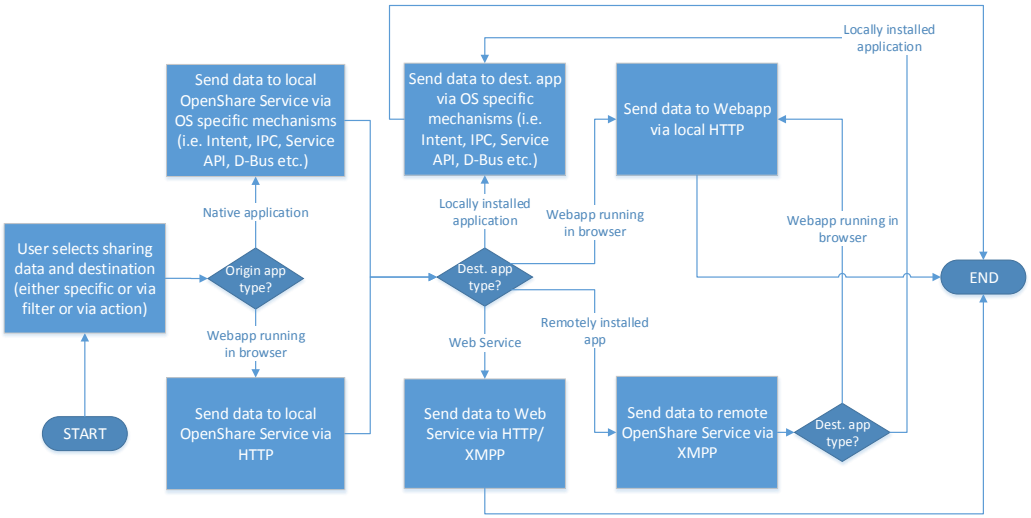

Fig. 5. System perspective: Internal view on the sharing workflow which involves local protocols along with HTTP and XMPP

work on multi-platform support as well as on developer and community traction through libraries especially for web and desktop applications.

V. Conclusion

OpenShare has been designed as powerful and flexible instant data sharing framework which overcomes many limitations of existing approaches already deployed in practice and proposed by previous research. In this paper, we have motivated the raison d'être for OpenShare. It combines local media-type actions with service-oriented matchmaking based on weighted non-functional property matching and both peer-to-peer and client-server interactions between devices and services. We have further formalized sharing activities, presented a sharing framework architecture and discussed initial implementation parts which cover certain phases of the timeline of sharing activities. This work uses and extends existing prototypes for device discovery and messaging between devices to demonstrate a proof of concept. It is ongoing with regards to the integration into native desktops and working environments. We plan to conclude the implementation and perform scenario tests with smaller groups of users as follow-up work.

Acknowledgements

This work has received funding under project number 080949277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony.

References

[1] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich, "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," *Wireless Personal Communications*, vol. 21, no. 1, pp. 77–103, April 2002.

[2] N. Hassan, M. M. Rahman, P. Irani, and P. Graham, "Chucking: A One-Handed Document Sharing Technique," in *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II (INTERACT)*, September 2009, pp. 264–278, Cape Town, South Africa.

[3] N. Marquardt, T. Ballendat, S. Boring, S. Greenberg, and K. Hinckley, "Gradual engagement: facilitating information exchange between digital devices as a function of proximity," in *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces (ITS)*, November 2012, pp. 31–40, Cambridge, Massachusetts, USA.

[4] A. Sinitsyn, "A Synchronization Framework for Personal Mobile Servers," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom) Workshops*, March 2004, pp. 208–212, Orlando, Florida, USA.

[5] R. Reichle, M. Wagner, M. U. Khan, K. Geihs, J. Lorenzo, M. Valla, C. Fra, N. Paspallis, and G. A. Papadopoulos, "A Comprehensive Context Modeling Framework for Pervasive Computing Systems," in *Proceedings of the 8th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS)*, June 2008, pp. 281–295, Oslo, Norway.

[6] Dropbox Inc. (2013) Dropbox. [Online]. Available: https://www.dropbox.com/

[7] PushBullet Inc. (2013) PushBullet - Easily push to your Android devices. [Online]. Available: https://www.pushbullet.com/

[8] G. Gionis, H. Desruelle, D. Blomme, J. Lyle, S. Faily, and L. Bassbouss, "»Do we know each other or is it just our Devices?«: A Federated Context Model for Describing Social Activity Across Devices," in *W3C/PrimeLife Federated Social Web Europe Conference (FSW)*, June 2011, Berlin, Germany.

[9] J. Spillner, J. Schad, and S. Zepezauer, "Personal and Federated Cloud Management Cockpit," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 36, no. 1, p. 44, February 2013, dOI: 10.1515/pik-2012-0149.

[10] T. Van Cutsem, "AmbientTalk: modern actors for modern networks," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, ser. SPLASH '11 Workshops. New York, NY, USA: ACM, 2011, pp. 227–230. [Online]. Available: http://doi.acm.org/10.1145/2095050.2095085

[11] D. Schuster, R. Lübke, S. Bendel, T. Springer, and A. Schill, "Mobilis - Comprehensive Developer Support for Building Pervasive Social Computing Applications," in *Networked Systems Conference (NetSys/KiVS) - Communication Software Award Demo*, March 2013, Stuttgart, Germany (Demo).