Syncables: A Framework to Support Seamless Data Migration Across Multiple Platforms

Manas Tungare, Pardha S. Pyla, Miten Sampat, Manuel A. Pérez-Quiñones
Dept. of Computer Science and Center for Human-Computer Interaction
Virginia Tech, Blacksburg VA USA.
Email: {manas, ppyla, msampat, perez}@vt.edu

***Abstract*— There is inadequate support for *seamless task migration* when users use multiple devices to perform a single task. In this paper we present the Syncables framework, a software framework designed to alleviate the problem associated with task migration. The Syncables framework can be used to migrate task data and state information across platforms. In the paper we describe the goals, design and an initial implementation of Syncables. We highlight features, such as a consistent naming scheme, ability for applications to define their own arbitrary Syncable types and use of transcoders and filters. Syncables will help us develop 'continuous user interfaces' that automatically bridge task disconnects.**

I. Introduction

We live in an age of mobile computing, and interact, on a daily basis, with different platforms such as desktop computers, laptops, personal digital assistants (PDAs), cell phones, and others. We define an "information cluster" to be the set of all devices a user interacts with, in the course of his/her daily tasks. Each of the devices in an information cluster offers a unique set of affordances in terms of processing capabilities, storage capacities, mobility constraints, user interface metaphors, and application formats.

Traditionally, design practice and emphasis have always been on creating systems for a single platform with minimal consideration to the role each platform plays in the larger context of the user's tasks. However, users often perform the same tasks across multiple platforms. For example, in our preliminary investigations, we found that users often use a desktop and a notebook for similar tasks (e.g., word processing), but they take their work home on a notebook. Generally, two or more devices are used either simultaneously or one after the other to perform a single task.

Given that a user's task is not necessarily constrained to a particular platform, but that the data, applications, and interaction metaphors are created in isolation for a single platform, a number of interesting problems such as task disconnects arise. We discuss this in the next section.

II. Motivation and Problem Statement

Migrating a task from one device to another requires stopping work on the first device, transferring current data to the second device, opening and loading an assortment of applications to complement or replace the applications being used on the first device, and then restarting work on the original task. In this process, users trudge out USB key drives, remote desktop software, e-mail and network file storage in an attempt to be able to use multiple devices to accomplish their task. The end result is that users must perform additional steps to be able to resume their task in another device.

A. Missing Support For Seamless Data Migration

Software designed for a particular platform does not provide mechanisms for seamlessly transferring users' data to other platforms in a user's information cluster. This often forces people to adapt their workflow to try to keep the work products of an interaction available to themselves on another platform. Jones, Bruce and Dumais [1] found that people often emailed the URLs of the websites they visited or the 'favorites' list to themselves when they have the need to access it from a different location. This approach is often preferred over bookmarking at the browser level, since most bookmarks are tied to each individual browser. In this example, users of multiple platforms adapted their workflow to compensate for the lack of support of task and/or data migration. The available option (e.g. bookmarking) does not support migration, thus users included in their workflow emailing their work to themselves.

It is worth noting that this problem is much larger than just for bookmarks. We have observed that many mobile workers have opted to use a single computer, their laptop, instead of moving files back and forth between their office and their off-site locations. To minimize the disruption, users simply opt to 'carry their office with them' by keeping all of their data in a single device (laptop) and taking it with them. The advantages of performance provided by a desktop computer are thus neglected in favor of ease of use.

B. Task Disconnects

A direct consequence of missing support for data migration is a *task disconnect*. We define a *'task disconnect'* as *'the break in continuity that occurs due to the extra actions outside the task at hand that are necessary when a user attempts to accomplish a task using more than one device'* [2].

C. Data Formats and Platform Dependence

Currently, most applications are created for a particular platform with a native file system and format that does not allow easy migration to a different platform. There is a need to have a mechanism to free the user's data from a particular platform and format and make such details transparent to the user. However, it should be noted that in order for seamless data migration, it is not necessary that all data be consistent (at the byte level) across all the devices in an information cluster. It is often enough to be able to access the same information in a different format than the original. The differing form factors, storage capacities and computational capabilities of each device make byte-level consistency not only difficult to achieve, but also undesirable from a usability perspective. For example, a full-page document in Portable Document Format (PDF) is harder to read on a mobile phone because of the increased scrolling necessitated by the small screen. However, a plain-text version of the same document with font sizes and line-wraps more suited to a small screen is easier to read. Much research has been done on transcoding web content to make it available in other platforms [3]. Even commercial applications, like web browsers, support some form of transcoding of content to support mobile phones, e.g. Opera, include special rendering modes that reformat the content to better suit the physical form factor of the device they run on.

D. Implicit Assumption that All Data is Contained in Files

Another shortcoming of current operating systems and application software is their emphasis on treating all of a user's data as files. As Barreau et al. [4] stress, considering a user's data as consisting of just files is to take a very narrow look at the data. One should be able to migrate data without having to save to a file first. However, we are not aware of any frameworks that allow such rich treatment of users' data.

III. Related Work

A few studies have tried to address the problem of migrating tasks or applications over multiple platforms; most of them have focused primarily on the technological aspects of this problem. For example, Chu *et al.* take the approach of migrating an entire application to support seamless task roaming [5]. However, their approach has considerable latency during migration (interrupting the user's tasks sequence) and does not discuss the implications on the user's tasks and goals. Similarly, Bandelloni and Paternò [6] talk about user interaction with an application while moving from one device to another. Chhatpar and Pérez-Quiñones [7] call this migration "dialogue mobility".

Barreau *et al.* [4], in their seminal study on users' perception of files and file systems, concluded that users do not always think of their data in terms of a hierarchical structure. Often, the data need not be associated with a file on disk and could be temporary. They classify users' data as *ephemeral*, *working* and *archived* data [4]. Ephemeral information has a short shelf life that is usually "loosely" filed. Working information is frequently-used information that is directly relevant to the task at hand. Archived information is infrequently-accessed information which usually represents completed work.

There are standard protocols for transferring files of various encodings over networks, e.g. MIME [8]. However, all such approaches deal specifically with files. Nelson *et al.* [9] describe an approach that deals with digital library objects instead of files; however, this approach does not provide any translation or filtering services and is not designed for use in a multi-platform context. Other works such as SODA [10], Fedora [11], and CNRI [12] are essentially architectures and frameworks to allow the extensibility of document behaviors. The focus of all these works is primarily on portability of content, and diversity and complexity of rendition across them. None of these efforts focus on migrating the intra-task context to help mitigate the disruptive effects of task disconnects.

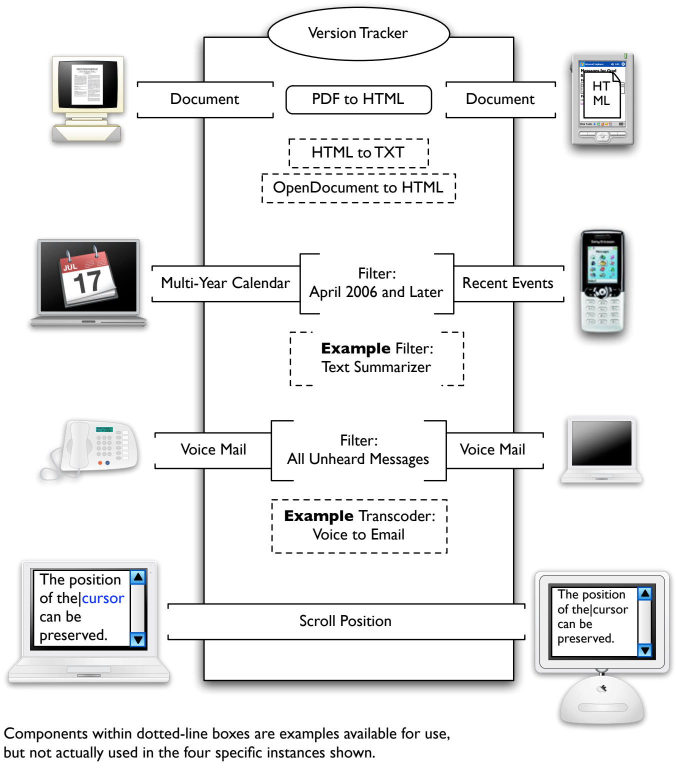
Version Tracker
Document
PDF to HTML
Document
HT ML
HTML to TXT
OpenDocument to HTML
JUL 17
Multi-Year Calendar
Filter: April 2006 and Later
Recent Events
Example Filter: Text Summarizer
Voice Mail
Filter: All Unheard Messages
Voice Mail
Example Transcoder: Voice to Email
The position of the|cursor can be preserved.
Scroll Position
The position of the|cursor can be preserved.
Components within dotted-line boxes are examples available for use, but not actually used in the four specific instances shown.

Fig. 1. The Syncables Framework

IV. Architecture and Implementation

The following sections describe the architecture of our Syncables system. The Syncables framework allows the synchronization of arbitrary objects, which are not necessarily files. The needs of every multi-platform application are different; thus, a one-size-fits-all solution is seldom appropriate. Including support for every possible scenario in a closely-coupled framework results in a code base that has high memory footprint and low adaptability. Instead, we take a different approach, where the framework provides a few basic services and the ability for applications to compose them together. Standard filters and transcoders are provided, and applications may extend the framework by defining their own synchronizable data types, data filters, and transcoders for various formats. This also addresses concerns of limited storage space and memory on portable devices such as PDAs and cellphones.

A. User-Defined Information Clusters

Our framework allows users to define and organize their information in terms of an information clusters. An information cluster is a grouping of their personal devices and the data there contained under a unifying logical organization. Some examples of information clusters can be found in IV-D.2.

B. Support for Collections

It is often necessary to synchronize only specific, well-demarcated parts of one's information cluster among devices. For example, a user may want all the information related to a particular project (files, email messages and bookmarks) to be synchronized between home and work machines, but prevent personal financial data from being shared with other machines. We refer to each such grouping of information objects as a *'collection'*.

The concept of collections helps bring together assorted data types such as files and email messages into a single hierarchy grouped by project, thus mitigating the problem of information fragmentation [13] and multiple redundant hierarchies [14]. We expect this approach to make it easier to manage personal information across a user's many devices.

C. Syncable Objects

A collection is composed of Syncable objects. A type of Syncable object is defined by each software application that wants to use the framework. Examples of Syncable object types include files, calendar events, music, email messages, notes, bookmarks, contact information, etc. A Syncable object must have three basic properties:

- **Name:** Each syncable object must be identified with a name. The fully-qualified name of the object, formatted as a URI, must be unique. The fully-qualified name is described in section IV-D.
- **OutputStream:** Whenever the data belonging to a Syncable object needs to be updated, the framework will write it to an `OutputStream` defined by the object. The object is responsible for determining where the data is finally written to. For example, a `File` Syncable object would pipe the received data to a disk, while a `Calendar` object would update the data in the user's calendaring program.
- **InputStream:** Similarly, the framework will read data from the object via an `InputStream`. The Syncables framework can only access data through this abstract stream; it is not aware of the physical source of the data, which makes the framework very generic.

These are the only three properties required of an object to enable synchronization using our framework.

D. Naming Scheme

When a unit of data is migrated among various devices, the exact location of that data should be device-independent and transparent to the user. In other words, whenever a request is made for that data by any application, it need not specifically state what device it is to be fetched from. The other primary motivation for such a naming scheme is the fact that each operating system or platform has its own conventions for naming archived files and where they are stored, while there is a conspicuous lack of naming conventions for working data or ephemeral data. With Syncables, our aim was to name each task information object uniquely across platforms, thus letting applications on different platforms share common ground about the data they work with.

Identification of a syncable object is via a Uniform Resource Identifier (URI) [15]. Each such syncable is assigned a URI that consists of five main parts, as illustrated in the example below:

`sync://`***`<info-cluster-id>/<collection>/<type>/<path>/<object-name>`***

1) Information Cluster ID: Each information cluster is identified by a Globally Unique IDentifier (GUID), a 128-bit number that is generated by an algorithm that minimizes the chances of any two of them being the same. Using a Cluster ID as part of the name of a syncable object makes it possible to use the same name for that object on all devices within the cluster. The framework (or, more specifically, the syncable type definitions) will take care of translating between local names and the syncable URI, just like a web server translates between disk paths and Web URLs.

2) Collection Name: This is simply a user-defined name given to the collection that the object is a part of. Typical examples include "Project X", or "Finances". It is worth re-stating that a collection can include information from multiple applications. For example, "Project X" would include files, calendar dates, emails, bookmarks, etc.

3) Syncable Type: This is a string that indicates the type of syncable object. Types are defined by each application that uses the Syncables framework. Examples include: `File` or a `Calendar` or a `Note`.

4) Path: The rest of the path is defined by the syncable type definition, and thus may be arbitrary. The examples below illustrate the genericity of the path that is possible.

5) Object Name: The specific object name is also assigned by the type, and the application treats that part of the URI as completely opaque.

Here are a few examples of syncable object names:

- `sync://`*`<info-cluster-id>`*`/IEEE-Portables /File/Research/Papers/Syncables.tex`

This is a syncable object of type *File*, in the collection *IEEE-Portables*, with the rest of the path consistent with the underlying disk path.

- `sync://`*`<info-cluster-id>`*`/IEEE-Portable /Email/Perez/Draft/3`

This is an object of type *Email*, belonging to the same collection as above, *IEEE-Portable*, included in the same hierarchy for easy retrieval.

- `sync://`*`<info-cluster-id>`*`/IEEE-Portable /Calendar/2006/09/01/Meeting-With-Manuel`

A *Calendar* event is part of the same project.

- `sync://<info-cluster-id>/IEEE-Portable/Note/Phone/Reminders`

Notes and other non-file objects can be easily accommodated by our framework.

E. Version Tracking

Any system that deals with data migration must address issues related to synchronization of data. A requirement for automatic synchronization is a mechanism for detecting version information for all the objects involved. Version information is tracked by a *Version Tracker* component for each type of object. Versions are expressed simply as timestamps in ISO 8601 format. This makes them easily sortable, and the timestamp provides a deterministic way of detecting the latest version. Each syncable object also records the last version that it was edited from, making it easier to spot situations where two or more copies of the same object may have been concurrently edited (without having been synchronized first). In such cases, the framework requests user intervention to resolve the conflicting edits.

The framework can query the version tracker, for example, for "a list of all objects modified since 2006-02-04", to which the version tracker object returns a list of URIs. The application can then decide how it wants to act on the information thus obtained – it can choose to synchronize all the changes, or synchronize some, or defer the synchronization entirely.

The important issue of out-of-sync clocks between the synchronizing devices is handled by first synchronizing the individual clocks using the Network Time Protocol (NTP). If a device does not have internet connectivity, its clock may be synchronized by the other device involved in the migration.

It is interesting to note that there is no centralized store for version info, and each syncable type tracks versions for all objects of its type. This makes it possible for ephemeral data stores such as clipboard data or scroll-position context to implement their own mechanism for controlling and notifying the system of updates.

F. The Process of Data Migration

In our current implementation, one device in each cluster acts as the master, and runs an HTTP server which supports both, `GET` and `PUT` methods for bidirectional data transfer. Information about devices in the cluster is stored locally on each device, so they can initiate data migration. All syncable types available for synchronization are available in a registry (local to each device) so that the framework is aware of the types it can and cannot handle.

A device may query the server for updates to a particular collection, further filtered by object type. The internal mechanism of synchronization is extremely simple: the framework uses the `InputStream` and `OutputStream` provided by each object to propagate updates.

G. Enabling Transcoding and Filtering

Some data formats may only be interpreted by software that runs on a specific operating system. In such cases, we aim to preserve the content of the document and make it accessible on other platforms by the process of transcoding. Transcoders for some common formats are available as part of the framework, and applications may define their own. Standalone transcoders for Web content are already available [16].

Partitioning of data and functionality deals with how a program divides what functions and what data is most appropriate on each device. Having a desktop calendar application show the entire month as a first view with overview information for each day put on the screen simultaneously is reasonable. On a cell phone, it is more appropriate to show only today's activities. Filtering is needed when attempting to migrate task information across platforms with different capacities and architectures. In such a case, data may be filtered for appropriateness by means of predefined or application-defined filters.

We use the HTTP Header `Accept:` to indicate the format in which the destination requests content. When a request is made with an `Accept:` header that is different from the source data type, the framework locates a transcoder from a registry of available transcoders, and pipes the content of the outgoing stream through the transcoder. Since filtering of objects is dependent on the specific type, the framework relies on the type definition to determine if filtering should take place, and the parameters for the filter. For example, a request for calendar items that ends in `/2006/05/` might represent an *implicit request* to filter calendar events to the month of May 2006.

V. Examples of Usage Scenarios of the Syncables Framework

In Figure 1, we show four examples of the Syncables framework being used for data migration. The first is a simple migration of files between a desktop and a PDA; transcoding is automatically performed for those file types that the PDA cannot display in their native format. In the second example, a desktop calendar is migrated to a cellphone, filtered to events since April 2006. Other filters may include automatic text summarizers for long passages. The third example is used to migrate all unheard voice messages from a corporate telephony system to a user's email inbox. A filter excludes all messages that have already been listened to. Transcoding could be used at this step to convert voice to text. The last example shows how application context information can be migrated across devices with dissimilar form factors and applications – in this case, the current scroll position and cursor position are made available to the second device which helps bridge the task disconnect.

VI. Future Work

We are currently creating many more Syncable object definitions, such as the ability to capture and relay information about running applications, documents, and contextual information about these applications. We intend to evaluate the framework from two perspectives: users' perspective and developers' perspective. In addition to satisfying our user goals, the framework must also be easy to use for developers. Since the usefulness of such a tool can only be gauged by repeated, frequent use in a mobile setting, we plan to conduct a longer-term field trial by inviting participants to try it for a significant amount of time. Information gathered from interviews and data collected from system logs can help us determine whether the framework represents a significant step towards bridging task disconnects.

In cases where only a small part of a large object has undergone changes, it is wasteful of network bandwidth to transfer the entire data again. We plan to use the `rsync` protocol which optimizes this transfer by sending only the changed bytes (plus some overhead) over the network.

In a future version, we aim to remove the requirement for one machine to act as the master. Instead, we plan to make data migration possible between any two devices in a user's information cluster.

VII. Conclusion

In this paper, we described the design goals and architecture of a framework for supporting seamless task migration across multiple platforms. The Syncables system is designed to work with multiple types of data that need not necessarily be saved to disk first, and also allows task information such as scroll position and other application context to be migrated from one platform to another. Recognizing that multiple platforms are often of varying form factors, the Syncables architecture allows complete extensibility for applications to create and compose filters and transcoders according to the needs of the specific application.

References

[1] W. Jones, H. Bruce, and S. Dumais, "Keeping found things found on the web," in *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2001, pp. 119–126.

[2] P. S. Pyla, M. Tungare, and M. Pérez-Quiñones, "Multiple User Interfaces: Why Consistency is Not Everything, and Seamless Task Migration is Key." in *Proceedings of the CHI 2006 Workshop on The Many Faces of Consistency in Cross-Platform Design.*, 2006.

[3] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd, "Power browser: Efficient web browsing for pdas," in *SIGCHI conference on Human factors in computing systems*, The Hague, The Netherlands, 2000, pp. 430–437.

[4] D. Barreau and B. A. Nardi, "Finding and reminding: file organization from the desktop," *SIGCHI Bull.*, vol. 27, no. 3, pp. 39–43, 1995.

[5] H.-h. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri, "Roam, a seamless application framework," *Journal of Systems and Software*, vol. 69, no. 3, pp. 209–226, 2004.

[6] R. Bandelloni and F. Paternò, "Flexible Interface Migration," in *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*. New York, NY, USA: ACM Press, 2004, pp. 148–155.

[7] C. Chhatpar and M. Pérez-Quiñones, "Dialogue mobility across devices," in *ACM Southeast Conference (ACMSE)*, Savannah, Georgia, 2003.

[8] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME)," http://www.ietf.org/rfc/rfc2045.txt, 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2045.txt

[9] M. L. Nelson and K. Maly, "Buckets: Smart objects for digital libraries," *Commun. ACM*, vol. 44, no. 5, pp. 60–62, 2001.

[10] M. L. Nelson, K. Maly, D. R. Croom Jr., and S. W. Robbins, "Metadata and Buckets in the Smart Object, Dumb Archive (SODA) Model," in *Third IEEE Meta-Data Conference*, 1999.

[11] S. Payette and C. Lagoze, "Flexible and Extensible Digital Object and Repository Architecture (FEDORA)," in *Second European Conference on Research and Advanced Technology for Digital Libraries*, Heraklion, Crete, 1998.

[12] W. Y. Arms, C. Blanchi, and E. A. Overly, "An architecture for information in digital libraries," *D-Lib Magazine*, February 1997.

[13] O. Bergman, R. Beyth-Marom, and R. Nachmias, "The project fragmentation problem in personal information management," in *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. New York, NY, USA: ACM Press, 2006, pp. 271–274.

[14] R. Boardman, R. Spence, and M. A. Sasse, "Too many hierarchies?: The daily struggle for control of the workspace," in *Proc. HCI International 2003*, 2003.

[15] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," http://www.ietf.org/rfc/rfc2396.txt, August 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2396.txt

[16] Z. Shao, R. Capra, and M. A. Pérez-Quiñones, "Transcoding HTML to VoiceXML using annotation," in *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence, 2003.*, 2003, pp. 249–258.