Syncables: A Framework to Support Seamless Task 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.
{manas, ppyla, msampat, perez}@vt.edu

ABSTRACT

When users use multiple devices to perform a single task, either simultaneously or one after the other, there is currently inadequate support for seamless task migration. We describe the goals, design and implementation of the Syncables framework that can be used to migrate task data and state information across platforms. Features such as a consistent naming scheme, ability for applications to define their own arbitrary Syncable types and use of transcoders and filters would help us develop 'continuous user interfaces' that automatically bridge task disconnects.

Categories and Subject Descriptors

H5.m [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous

1. INTRODUCTION AND MOTIVATION

In this age of mobile computing, it is extremely common for users to perform their tasks using multiple devices such as desktop computers, laptops, personal digital assistants (PDAs), cell phones, and others. Two or more of these devices are often used either simultaneously or one after the other to perform a single task. However, migrating a task from one device to another requires stopping work, 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. Context information is often lost, and users must perform additional steps to be able to resume their task where they left off. 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'*.

One of the main issues that contributes to the high cost of task disconnects is that the by-products of interaction, (i.e., files, notes, bookmarks, scroll position within a document, etc.) are not automatically made available to the user when resuming a task. In this paper, we describe the design goals and architecture of a framework that applications can use to migrate their tasks seamlessly across devices. The framework also provides features for transcoding data so that it may be readable on a different, less-capable platform, as well as services to filter data for appropriateness to a particular platform. We present the design goals, architecture, implementation aspects, and prototype applications that use the Syncables framework to enable seamless migration of application data, files, and context.

2. RELATED WORK

Barreau *et al.* [2] 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 [2]. 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. Current file systems help manage only archived data, and it is possible to copy such archived data between platforms by various means (e.g. via disks or USB drives, by sending it via a network, or by attaching files to an email.)

Being able to transfer files of various encodings [6] over standard protocols is a problem dealt with in various scenarios. However, all such approaches deal specifically with files. An approach that deals with digital library objects rather than files is described in [9] where document metadata is encoded within an object, so that it may be used with any digital library collection. However, this approach does not provide any translation or filtering services and is not designed for use in a multi-platform context.

Multivalent documents (MVD) [12] extend document 'behaviors' to allow portability across platforms as well as applications. However, that work is focused on annotation and portable layers atop the original document, and the effects of multiple devices were not explored. Similarly Dushay [5] introduces the concept of using context brokers as information intermediaries to allow designers to design different experiences for the same document content. Our system, in spite of performing such functions, is different in that it actively addresses seamless task migration. Other works such as SODA [10], Fedora [11], and CNRI [1] 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.

3. SEAMLESS TASK MIGRATION ACROSS MULTIPLE PLATFORMS

Our overarching research objective in this work is to create a framework that seamlessly migrates a task across different platforms. Migrating a task across platforms entails many dimensions such as the transfer of data and context, the “manipulation” of data to suit the target platform, the provision of cues to recover context on the second platform, etc. In the interest of space, a complete discussion of all the dimensions in seamless task migration is not possible here.

4. SCENARIO

To explain the problem with more clarity, we present a scenario: Amy is a financial advisor who must constantly be aware of the latest information about the technology sector, and be able to make informed decisions on when to sell or buy particular stocks. She monitors financial news on various financial news channels on a television at her office, uses a word-processing program and a spreadsheet on her desktop computer, and specialized software to manage stock portfolios for her clients. When meeting clients at their locations (i.e. outside her office), she is forced to transfer manually all the related documents to her laptop. A mistake can prove very costly, since the data is time-sensitive. If she intends to continue working on her task on her laptop, she must note the current position within the spreadsheet on the desktop, save the document to a USB drive, and then reopen the document on her laptop. Contextual information such as position within the spreadsheet is not available to her automatically on her laptop with current systems and methods.

Amy also needs to access news and client portfolios using her PDA – but she can only do that if she has remembered to synchronize her PDA with her desktop before she left her office. She is currently unable to view on her PDA any financial reports that have a graph or chart within them, because the software on her PDA cannot interpret those items within a document (thus rendering the entire document unreadable).

With Syncables, Amy no longer needs to explicitly synchronize her devices before leaving the office. Since the PDA software cannot view graphs or charts, the reports are automatically converted to a temporary format without the graphs or charts, so that Amy can at least read the text matter from them. She also has access to the last week of activity on her clients’ portfolios via her cellphone (though not the entire history, since cellphone memory is limited as compared to desktop or laptop computers.)

5. DESIGN GOALS

The following sections describe the design goals we pursued when creating the Syncables framework.

5.1 Multiple Information Clusters

We define an *‘information cluster’* as the set of devices among which tasks need to be migrated. Thus, a typical information cluster would consist of all the devices owned by a particular user – home desktop, work desktop, laptop, PDA, cellphone, etc. In order for seamless task migration, it is not necessary that all data be consistent across all the devices in an information cluster; i.e. perfect data consistency is not an aim. The differing form factors, storage capacities and computational capabilities of each device obviously make this requirement impossible.

An information cluster need not be bound to a user: several users may elect to form an arbitrary information cluster out of their devices for collaboration purposes. For example, members of a research group may choose to form an information cluster that includes their laptops for sharing papers they read, or documents they are working on. Users can define the specific task information from their devices that should be migrated, thus separating private and public files and their migration in different information clusters.

5.2 Not Just Files

As Barreau *et al.* [2] stress, considering a user’s data as consisting of just files is to take a very narrow look at the data. Our goal was to enable migration of ephemeral, working as well as archived data. To be able to migrate a unit of data across devices, it need not be saved to a file first.

5.3 Consistent, Hierarchical Naming

When a unit of data is synchronized 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.

Although the names assigned to task information in our framework need not be seen or used by users directly, there is value in maintaining a hierarchical structure in the naming scheme [8]. Instead of separate address spaces for filenames, email, bookmarks, and other task information, a single naming scheme for all information items addresses the issue of information fragmentation [7] and project fragmentation [3].

5.4 Use of Standard Protocols

With various standard protocols available for data transfer, we wanted to avoid reinventing the wheel, and make use of these protocols for several reasons. Firstly, standards are well-documented, thus it helps other developers interested in extending the system. Secondly, some devices have limited programmability (e.g. some cellphones), but certain standard protocols (e.g. HTTP) are implemented on those devices. Applications can thus use these pre-implemented protocols where custom protocols are impossible to program. Thirdly, it is easier (from an implementation point of view) to assemble a system from well-tested open-source code that handles these standard protocols. Finally, using open standards allows for creation of interoperable systems that may each be written using the language/toolkit that best suits the platform in question.

5.5 Support for Transcoding and Filtering Data

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.

Filtering is needed when attempting to migrate task information across platforms with different capacities and architectures. For example, a calendar application on a desktop may retain events from the past several years; however, a calendar application on a device such as a phone or PDA typically does not need to have the entire calendar data available. Often, the events for the current week or current month are sufficient to satisfy the demands of the mobile user. In such a case, data may be filtered for appropriateness by means of predefined or application-defined filters.

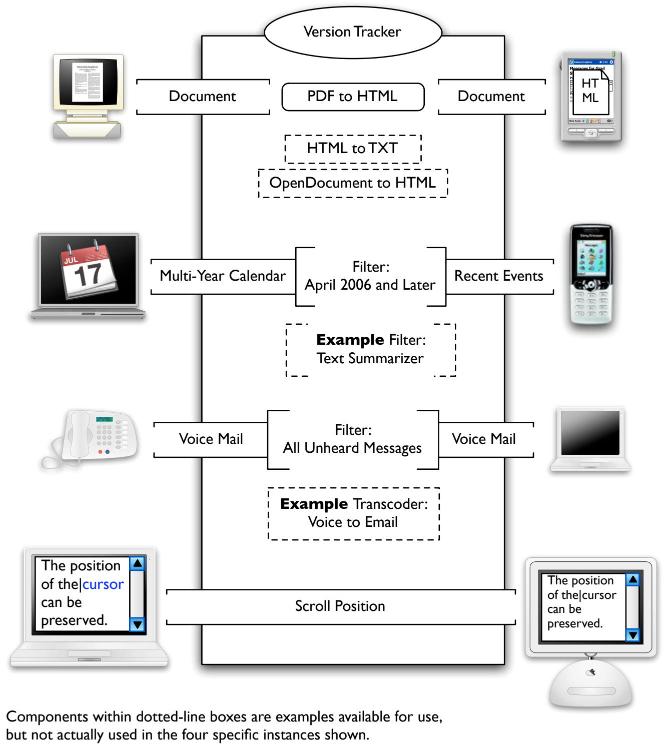
Version Tracker
Document
PDF to HTML
Document
HT ML
HTML to TXT
OpenDocument to HTML
JUL 17
Multi-Year Calendar
Filter: April 2006 and Later
Recent Events
Example Filter: Text Summarizer
Voice Mail
Filter: All Unheard Messages
Voice Mail
Example Transcoder: Voice to Email
The position of the|cursor can 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.

Figure 1: The Syncables Framework

5.6 Graceful Degradation of Content Format

On platforms with limited features, it may not be possible to render content as originally intended. For example, a PDA may lack the software and computation power needed to display documents with complex formatting. However, there is some value in being able to see the content, minus the formatting, even if the full richness of the document cannot be viewed. Lossy transcoders may be used for such conversion. Other types of content may also need to be downgraded to a potentially lossy format.

5.7 Complete Extensibility

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.

6. ARCHITECTURE AND IMPLEMENTATION

The following sections describe the architecture of our Syncables system, as shown in figure 1. A prototype of this system has been developed in Java; although some terminology in the following sections may refer specifically to Java, it is generic enough to be re-implemented in any language.

6.1 Syncable Objects

Any application that wishes to synchronize data must define its Syncable objects (or use any of the pre-defined ones). A few examples of Syncable objects would be files, calendar events, email messages, notes, to-do items, contact information, etc. Not only information that is typically stored on a computer, but also information such as the current television channel, or radio station frequency, or the list of unheard voice-mail messages can be a syncable object in our framework.

A syncable object may consist of any application-defined data, but must expose three properties to the framework. (This is enforced in our implementation by having all syncable objects conform to the `Syncable` interface. Each syncable object must be identified with a unique name, as described in section 6.2. Whenever data needs to be written to a syncable object, it is written to an `OutputStream` exposed by the object. Similarly, data may be read from the object via an `InputStream`. The framework does not try to interpret the resulting stream of bytes in any meaningful way (transcoders and filters do, but the framework itself does not assign any meaning to these bytes.) These are the only three necessary properties of any syncable object.

6.2 Naming Scheme

Identification of a syncable object is via a Uniform Resource Identifier (URI) [4]. Each such syncable is assigned a URI that consists of four main parts, as illustrated in the example below:

`sync://`***`<info-cluster-id>/<syncable-type>/`***
`<path-defined-by-type>/<object-name>`

6.2.1 Information Cluster ID

Each syncable object is migrated within a single information cluster. (If the same piece of data needs to be migrated among many information clusters, it will need to have multiple names which all refer to a (shallow) copy of the data.) An 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.

6.2.2 Syncable Type

This is a string that indicates the type of syncable object: it may be a `File` or a `Calendar` or `Note`, or a type custom-defined by an application. This ensures that data of a certain type is not misinterpreted as another type during task migration.

6.2.3 Path Defined by Type

The Syncables framework does not attempt to assign any meaning to the rest of the path, and the rest of the hierarchical name of the object is thus entirely defined by an application.

6.2.4 Object Name

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

6.2.5 Examples

The first two examples are file paths, while the last two represent an arbitrary hierarchical structure.

- sync://1D220FFE-B291-58B1-FAA1-C96B7883225C /File/Documents/Research/MobileHCI/Syncables.tex
- sync://1D220FFE-B291-58B1-FAA1-C96B7883225C /File/Music/ABBA/Gold/Chiquitita.mp3
- sync://1D220FFE-B291-58B1-FAA1-C96B7883225C /Calendar/2006/05/01/Meeting-With-Steve
- sync://1D220FFE-B291-58B1-FAA1-C96B7883225C

/Note/Phone-Reminders

6.3 Version Tracking

Each syncable type (not each object) has an associated mechanism for tracking object versions. 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 synchronization).

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. Interestingly, this clock synchronization also occurs using `Time` syncable objects.

The version tracker object for each syncable type internally maintains version information about objects of its type. The framework queries 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 is interesting to note that there is no centralized store 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.

6.4 The Task Migration Process

The power of the Syncables framework is its simplicity: the framework makes a stream of data available from one device to another and lets applications migrate tasks. In our current implementation, each device implements a simple HTTP stack, and is able to transfer data as a stream of bytes. In cases where only a small part of a large file has undergone changes, it is wasteful of network bandwidth to transfer the entire file again. In a future implementation, we plan to use the `rsync` protocol, which optimizes this transfer by sending only the changed bytes (and some overhead) over the network. It is crucial to note that the choice of underlying transport mechanism is independent of our design decision to use URIs for internal naming of syncable objects.

6.5 Enabling Transcoding and Filtering

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. The output of the transcoder is relayed to the destination. 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 a request to filter calendar events to the month of May 2006.

7. 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.

8. CONCLUSION

In this paper, we described the design goals and architecture of an environment 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.

9. REFERENCES

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

[2] D. Barreau and B. A. Nardi. Finding and reminding: file organization from the desktop. *SIGCHI Bull.*, 27(3):39–43, 1995.

[3] 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*, pages 271–274, New York, NY, USA, 2006. ACM Press.

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

[5] N. Dushay. Localizing Experience of Digital Content via Structural Metadata. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 244–252, New York, NY, USA, 2002. ACM Press.

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

[7] W. Jones and H. Bruce. A Report on the NSF-Sponsored Workshop on Personal Information Management, 2005.

[8] W. Jones, A. J. Phuwanartnurak, R. Gill, and H. Bruce. Don't Take My Folders Away!: Organizing Personal Information to Get Things Done. In *CHI '05: CHI '05 Extended Abstracts on Human Factors in Computing Systems*, pages 1505–1508, New York, NY, USA, 2005. ACM Press.

[9] M. L. Nelson and K. Maly. Buckets: Smart objects for digital libraries. *Commun. ACM*, 44(5):60–62, 2001.

[10] M. L. Nelson, K. Maly, D. R. C. 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] T. A. Phelps and R. Wilensky. Multivalent documents. *Commun. ACM*, 43(6):82–90, 2000.