Semantic-Chunks
A Middleware for Ubiquitous Cooperative Work

Luís Veiga and Paulo Ferreira
INESC-ID/IST, Distributed Systems Group, Rua Alves Redol N. 9,1000-029 Lisboa, Portugal
{luis.veigal|paulo.ferreira}@inesc-id.pt

ABSTRACT

To be productive, cooperative work has to be supported efficiently so that users do achieve their goals. This requires solving the well-known fundamental problem of replicas consistency.

Update-based solutions are easy to use transparently with commercial applications, but consider every modification in a document as a new document update, thus fostering conflicts and hindering concurrency. *Operational-based* solutions promise increased concurrency, by interleaving compatible modifications from different users. They require central reconciliation algorithms, and cannot be applied to commercial applications without further instrumentation.

We propose the notion of a semantic chunk, i.e., a semantically-annotated document region with application relevance, that is promoted to a full-right entity w.r.t. consistency information and enforcement. This unit, being smaller than a file and semantically richer, allows greater concurrency and better update merging with less aborts than current solutions.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications

Keywords

Replication, mobility, file systems, office applications, consistency

1. INTRODUCTION

Information sharing is a fundamental aspect to computer supported cooperative work (CSCW) [9], and has been one of the main goals of distributed systems research. This has become even more so, recently, in the related fields of mobile, pervasive and ubiquitous computing. More and more people perform work and exchange data using their laptops, PDAs or mobile phones, even without being connected to a central network (e.g., using Bluetooth). In this context, data-replication has been a prime technique used for information sharing. It improves availability, performance, and cost-effectiveness.

Locally replicated data is always readily available to applications (even when the network is down), with access times orders of magnitude lower than non-local data, and avoids frequent, and possibly lengthy and costly connections to the underlying network (specially so in the case mobile devices with GSM, CDMA or GPRS connectivity).

Data replication has been comprehensively addressed in various projects and systems where applications are based either on files, databases, objects, application components, and structured documents (refer to [8] for a detailed review).

RM'05, November 28-December 2, 2005 Grenoble, France

Since several (or all) of the replicas may be independently updated (accessed and modified), the issue of replica consistency naturally arises. Traditional pessimistic approaches are based on locking resources. They are not suited for ubiquitous computing since applications, possibly disconnected from the network, perform long duration data manipulations.

Optimistic approaches allow concurrent modifications on different replicas, in the expectation that conflicts will never or seldom occur. If there are any, the system will attempt at reconciling them later. This advantage is even more relevant in pervasive and ubiquitous computing since applications are seldom connected, and even when they are, it may be impossible to access a specific machine (e.g., a central server), like in the case of a spontaneous ad-hoc network.

Data consistency enforcement and update reconciliation techniques include: i) determining causality among updates performed on different replicas and epidemically propagated [21, 2] among different peers; and ii) replaying of logged operations performed by different client nodes [12, 19].

Leveraging knowledge from application, data and usage semantics is a useful technique to improve replication management and consistency enforcement. It helps reducing the number (and cost) of update conflicts, and cuts down the amount of user's work lost when reconciliation of conflicting updates is impossible.

Semantics, w.r.t consistency, has been previously addressed by: i) using type and application-specific reconciliation procedures (opaque, i.e., non-transparent to the middleware) invoked when a conflict is detected, that process the conflicting updates and decide which update will prevail [21]; and ii) using application activity (operations, logged or inferred) and their semantic constraints (declared or inferred) to centrally re-order (re-schedule) them in a way that minimizes conflicts [12, 19].

1.1 Shortcomings of Current Solutions

The issue of supporting ubiquitous cooperative work, in the context of collaborative document edition, involves roughly three kinds of problems: i) enforcing replica consistency in an environment with de-centralized replication, while favoring update concurrency and successful update commits and merging; ii) allowing the users to use the same off-the-shelf office applications they already do, while guaranteeing flexible inter-operability between these and the replication and consistency systems; and iii) reduce wasted storage and network bandwidth. Although these sub-problems have been addressed (all or just one) in previous work, we propose a novel approach that, we argue, is better suited for abovementioned activities.

There are the two main families of approaches to consistency in mobile and ubiquitous environments. The first, *update-based*, is easy to integrate transparently with *everyday* commercial applications but considers every modification in a document, however small, as a new document update, thus fostering conflicts and hindering concurrency. The second, *operational-based*, promises increased concurrency by attempting to interleave compatible modifications from different users. However, it requires costly reconciliation algorithms, and has issues w.r.t. integration and inter-operability with

everyday commercial applications. Hence, it is difficult to apply to them, without further instrumentation. Merge procedures can be used in *update-base* systems to reduce conflicts but they are opaque to the rest of the middleware. They behave like black-boxes, difficult to port and reuse, and their code must be blindly trusted. They are inflexible, unadaptable w.r.t. having a clear interface with which the middleware could parameterize the conciliation task with additional information (e.g. gathered from context).

We try to establish a middle-ground between these two "opposing" approaches, avoiding the problems of each one, while retaining the advantages of both. From the *operational-based* approach, we take increased concurrency, but without the need to modify applications. From the *update-based* approach, we take transparency w.r.t applications, but attempt at reducing update conflicts.

To improve efficiency of cooperative work, application semantics must be considered w.r.t. data replication and consistency. Leveraging application semantics has been previously considered mainly for purposes of operation logging, and designing merge procedures. In this work, we make use of the structural and usage semantics of documents and applications (i.e., the emphasis is on the structure of the data that applications manipulate, and user behavior, regardless of the internal operations they perform). Therefore there is no need to log operations, just manage updates.

Techniques to exploit content similarities among different files, and especially, among versions of the same file, in order to save storage space and bandwidth, have been used prior in [14, 2].

1.2 Contribution and Paper Structure

We propose a novel approach to system support for cooperative work in ubiquitous environments, namely collaborative document edition. It is embodied in the notion of a semantic-chunk (semantic-regions in general, targeted to a chunk-based system [14]), i.e., a semantically-annotated document region with application relevance, that is promoted to a full-right entity w.r.t. consistency information and enforcement.

It fulfils the requirements of reduced memory and network usage. It provides the same level of optimistic consistency offered by current solutions, but with greater concurrency and flexibility, thus, less prone to update conflicts. It especially suited for, and easily lends itself to document edition applications. Finally, we describe how an adaptive middleware based on semantic-chunks can be integrated with popular office applications, using their restricted reflective capabilities (namely *automation*).

The rest of the paper is organized as follows. In the next Section, we present thoroughly the architecture of Semantic-Chunks w.r.t., successively: i) system overview, ii) storage and communication, iii) application data structure, iv) middleware for application enhancement, and v) consistency enforcement. Section 3 describes the main aspects regarding the proposed implementation, followed by Section 4, where we introduce some related work and projects. Section 5 wraps the paper up by offering some discussion, drawing conclusions and uncovering possible lines of future work.

2. ARCHITECTURE

We now briefly introduce some basic notions that are used throughout the rest of the paper, and will be explained more thoroughly in the next sub-sections. Chunks (or data-chunks) are portions of files that have content-derived (instead of offset-derived) boundaries, as introduced in LBFS [14]. A semantic-chunk manages a semantically relevant document region. The content of a semantic-chunk is an array of data-chunk IDs, thus referencing the data-chunks that actually hold the content of the document region.

The operating environment and usage model addressed in Semantic-Chunks is one of ubiquitous CSCW, namely, ad-hoc network-based

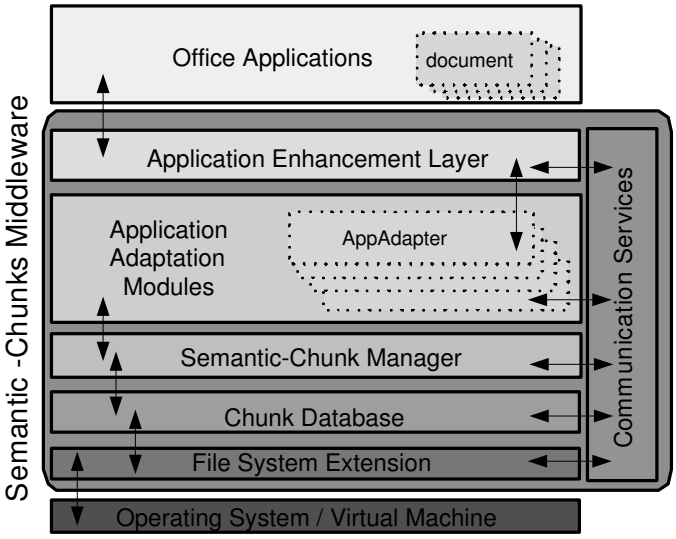

Figure 1: Semantic-Chunks System Architecture.

cooperative document edition (or cooperative document edition in ubiquitous computing). In this model, users make use of mobile devices (typically laptops and PDAs) to create, edit, and exchange documents, performing cooperative work based on office-like applications. Users access and (possibly) modify locally, documents that may have been either created locally, or replicated from another user's device. Users may also (re-)replicate a locally replicated document to another device.

We do not assume the presence of a fixed network infrastructure (either wired or wireless-based). Users must then rely on the limited wireless networking capabilities available on their devices. These are short-ranged, low-bandwidth (e.g., Bluetooth), further aggravated by the device's reduced battery life-time. Thus, users must engage preferably in spontaneous and short-duration ad-hoc network communication, in a variety of situations.

User activities include not only collaborative text edition (that receives the primary focus in the paper) but also other popular office-like applications like the edition of spreadsheets, web content, slide-based presentations, etc. In the remainder of this paper, we will use the term *document* as an abbreviation for *structured document*. Both the above mentioned types of documents may be encompassed by this notion, with the necessary and relevant adaptations.

2.1 Overview of a Semantic-Chunks System

The system architecture of Semantic-Chunks is depicted in Figure 1. It is is not bound to a specific platform or set of applications, since we intend it to be applicable generically. Nonetheless, we present a typical implementation in Section 3. Users operate essentially unmodified office applications, extended by an adaptable middleware layer that makes use of applications' reflective capabilities, namely *automation*.

Semantic-Chunks middleware mediates the applications and the operating system or virtual machine environment they run on. File System Extension provides a (virtual) file system abstraction for users to create, copy, delete files on (virtual) folders managed by Semantic-Chunks in their device, or on nearing devices also enabled with Semantic-Chunks. The Chunk Database stores chunks, i.e., variable-sized fragments of document data. The Semantic-Chunk Manager aggregates one or more chunks of data (as needed) that comprise logical units of documents, according to application semantics (e.g., sentences, paragraphs, sections, spreadsheet and slide regions, etc.). Semantic-chunks, themselves, are also aggregated in

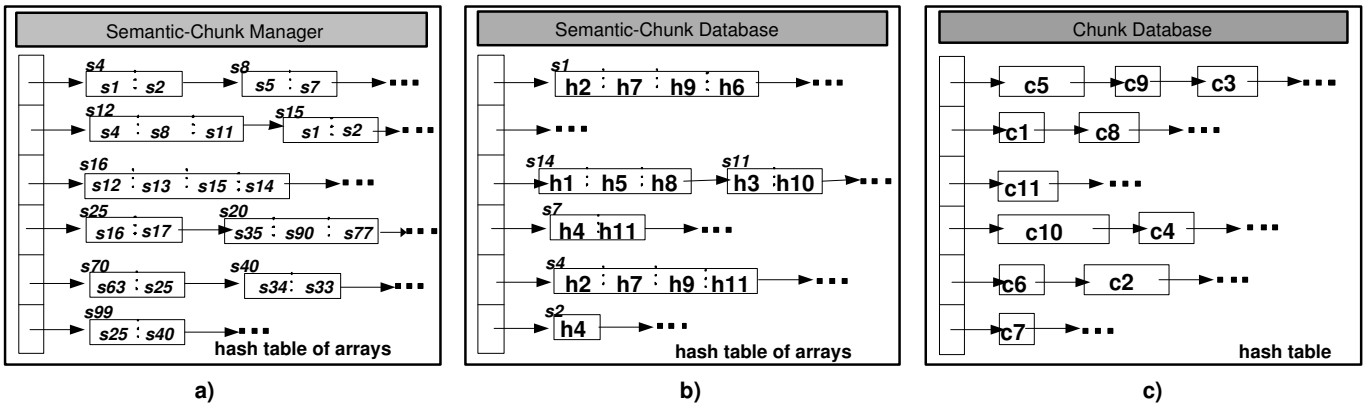

Figure 2: Logical Architecture of Storage in a Semantic-Chunks System.

higher-level semantic-chunks. A top-level semantic-chunks is the entry-level of a structured document. A large-sized document content can be incrementally fetched, and incrementally rendered, as users need it, or as updates arrive.

Whenever the office applications need to retrieve data from (or write to) the file system, those calls are intercepted by the File System Extension and it instructs the Application Enhancement Layer to interface with the office application (via its reflective capabilities) in order to fetch from (or inject in) it content (according to read or write operations). This content is, conversely, structurally injected back (or fetched from) to the Semantic-Chunks Manager (and hence, in the Chunks Database). This *loop-back* approach is key to maintain application transparency. To address the specifics of each application, basic mediation is extended by specific Application Adaptation Modules.

Communication with other mobile devices (in order to replicate files, propagate updates, etc.) is performed through a Web-Services based Communication Services bridge. Interaction with other devices can be performed at a variety of levels, as portrayed in Figure 1. These interactions may be exchanging (for either propagation or update purposes) of basic data-chunks, incrementally reconstructing semantic-chunks, downloading Application Adapters, etc.

2.2 Basic Storage and Communication

PDAs are devices with several resource constrains. Storage, bandwidth, processing power and battery life, are of premium importance. In the context of this work, we address solely storage and bandwidth limitations.

PDAs have no real (hard-)disks. Thus, there is no mass-memory support, since they are usually equipped with memory that is limited in size (normally, 32 or 64 MB). Therefore, file system is actually simulated in the PDA's memory. Although, there are Flash-memory cards (CompactFlash, SecureDigital, etc.) extensions ranging to and over 1 GB. However, these capacities are not common in most PDAs.

To address storage limitations, information redundancy must be exploited without incurring in high performance penalties (that could overload the processor and drain battery faster, as well). Data compression *per se*, besides performance demands, is not incremental i.e., even small changes in the beginning, middle, or end of a document affect the whole of the compressed version (since file must be compressed again). Thus, data similarities among different versions of the same document, and even among different documents, are leveraged without compression.

The basic storage substrate is therefore provided by a Chunk Database, as presented in [14]. An example is presented in Figure 2-c. It is structured as a hash-table of chunks. Chunks are portions of files that have content-derived (instead of offset-derived) boundaries. This way, they are mostly invulnerable to insertions, deletions, or other modifications, occurring in other regions of the file (e.g., in the beginning of the file). Only the neighboring chunks might be affected.

Chunk boundaries are determined resorting to Rabin fingerprints[16]. Conceptually, file contents are scanned with an overlapping window (*48 bytes* in length), and this determines a binary polynomial representation of the data modulo another pre-determined polynomial (one that is irreducible). This operation is performed efficiently with a sliding-window. The space of fingerprint values for regions can be segmented in two (not necessarily contiguous) subsets, so that certain values trigger the creation of a chunk boundary.

Chunks are identified by a hash value of their contents, using SHA-1 [22] algorithm. Only the first 64 *bits* of the SHA-1 hash value are taken as a chunk hashing key. Thus, every chunk has a (just) 8 *byte* sized identification key, while its size may range hundreds of bytes. This is instrumental in reducing storage occupation and bandwidth usage, as we explain next.

Whenever two neighboring devices need to exchange document contents (previously divided in chunks), e.g., while replicating a document, they exchange first the chunk keys they hold (regarding that file) and transmit only the contents of the chunks they do not hold already, hence saving bandwidth.

Whenever a user copies a file, creates a new version of a file, or receives an update from another user, it is expectable that most of the file contents remain relatively similar (i.e., apart the changes performed). Similarity among files residing in local storage is thus leveraged because when the new version or file is saved, most of its contents are already in the Chunk Database. Therefore, instead of copying the whole file contents, most of time, it just creates additional references to existing chunks, hence saving storage.

2.3 Application Data Structure

Naturally, applications are unaware of chunks. This is needed since normally, they just see flat files (regardless of their internal format specific to the application). Although using chunks enables the reduction of storage and bandwidth usage, it does not allow extracting and leveraging any kind of semantic information, from the applications or documents.

To address this, each document is decomposed in semantically relevant regions i.e., document regions that are significant to the document structure, to application semantics, and to typical usage behavior. These semantic-regions are called semantic-chunks because of the underlying Chunk Database. They are defined hierarchically and are application-semantics dependent, following naturally from the hierarchical structure of office-like documents.

Application-based semantic-chunk borders may be defined as sec-

tions, paragraphs, sentences in text documents, cell areas in spreadsheets, objects and geometry in CAD tools, functions and declaration zones, in programming source code editing, etc. This management is easy for applications because they know best the data and its structure.

Figure 2-b depicts a Semantic-Chunk Database. It is a hash-table of *Level-1* semantic-chunks (with data-chunks being at *Level 0*). At *Level 1*, semantic-chunks are arrays of chunk-ID, referencing those chunks that actually hold the contents of the semantic-chunk. A chunk can be shared by multiple semantic-chunks, belonging to any number of files. A semantic-chunk may also be shared by multiple versions of the same file.

Figure 2-a depicts the Semantic-Chunk Manager, where higher-level semantic-chunks are stored. The content of a higher-level semantic-chunk is an array comprised of IDs of other semantic-chunks. This structure allows an hierarchy with arbitrary number of levels. A file, besides other information regarding file system attributes, is simply regarded as a top-level semantic-chunk. Semantic-Chunks are identified by a GUID based on document name and device of creation.

Conceptually, to search for a specific semantic-chunk, the Semantic-Chunk Manager must be queried first, to check if it is a high-level semantic-chunk. If not found, then the Semantic-Chunk Database must be queried. In practice, all semantic-chunks are fetched from the Semantic-Chunks Manager, and whether it belongs to *Level 0* or is a higher-level semantic-chunk, a pointer to the appropriate array is returned, along with level info.

Semantic-chunks are transmitted either i) as a sequence of chunks (much as files in LBFS, or ii) as a sequence of other (lower-level) semantic-chunks. The hierarchy of semantic-chunks in each document is XML-described, whenever it is transmitted, as part of a web-service invocation. Each semantic-chunk has associated with it meta-data, namely regarding consistency.

Thus, in Semantic-Chunks (analogously to LBFS chunks), when two neighboring devices need to exchange document contents, they first exchange semantic-chunks, and then, if the receiver does not have them, they exchange the comprising lower-level semantic-chunks. This may trigger exchange of more semantic-chunks (new ones and others that may have been updated). Ultimately, exchanging new content may also cause transmission of data-chunks.

It is very frequent in collaborative edition of long-sized documents (i.e., several chapters or sections, regardless of actual file size) that some users only deal with a subset of fractions of the file (the ones they are collaborating in). Still, with current systems, every single one of them is compelled to store a copy of the complete file. In Semantic-Chunks, a document may be rendered without needing all of its lower-level semantic-chunks (and corresponding data-chunks) because, while the content is unknown, the document structure is already known, and the middleware can adapt accordingly.

2.4 Middleware and Application Enhancement

The main goal of the application enhancement layer is to enhance application functionality without changing application code (neither extending nor instrumenting it explicitly). This layer of the middleware has a number of responsibilities, broadly: i) manage document structure, ii) manage document content within Semantic-Chunks managed folders, and iii) manage exporting/importing documents to/from non-managed folders.

W.r.t. managing document structure, it is the responsibility of the application enhancement layer, through an adequate application adaptation module, to detect the basic structure of a document once it is inserted in a folder managed by Semantic-Chunks.

Semantic-chunk division must be performed with criteria in order to minimize overhead due to increased number of higher-level semantic-chunks. Thus, documents with reduced content and size are divided in fewer levels than documents larger sized. Larger documents can accommodate more semantic-chunks and more hierarchical levels, without significant penalties, comparatively, in terms of storage.

To allow incremental content replication, in order to save storage, the system must be able to replicate semantic-chunks as they are needed by the application, requested by the user, and accommodate them as they arrive. The application enhancement layer is in charge of inserting stub content (possibly with scripting to trigger download content) to replace, w.r.t. rendering, document regions that are missing, while the comprised semantic-chunks and data-chunks holding their content are still not available.

The application enhancement layer is in charge of mediating the export of document content (extraction) to corresponding semantic-chunks and, for each semantic-chunk render its contents. The Semantic-Chunk Manager will then store it, possibly updating other semantic-chunks, and associate it with consistency information. A converse procedure is performed to mediate the import of document content (injection) from semantic-chunks to be rendered in screen.

This layer is also responsible for importing and exporting documents from/to directories that are not managed by Semantic-Chunks. It handles the specifics of converting documents to and from their native format. These tasks can also be performed resorting to *automation*.

The adaptability of the middleware stems from the fact that it can be automatically adapted to other applications and their formats. This is performed resorting to Communication Services downloading new plug-able application adaptation modules, either from fixed network or from another device. This may be performed upfront or triggered when a folder managed by Semantic-Chunks receives a file of a format yet unknown. Type identification is still performed solely based on file name extension (.doc, .xls, etc.).

Replication and consistency sub-systems are unaware of the different semantic-chunks representations, so the base system is kept unchanged as new types of document formats are introduced. It is up to the application extensions, in a well defined cooperation with the storage and propagation system, to ultimately decide the actual content organization that will be presented to users. In this way the system is open and extensible, driven by an adaptive middleware layer and leveraging application reflective capabilities (even if limited). By exploiting this semantic knowledge about data structure and typical usage, in a transparent way w.r.t to the underlying semantic and data-chunk propagation system, favoring system modularity.

The usage of application automation API can be regarded as a form of reflective capabilities (even if limited). The code invoking this API gains, through them, access to the very structure and content (that may be changed) of the documents, and functionalities of the application itself. This portrays aspects of introspection and modification that comprise reflective capabilities.

2.5 Data Consistency

Chunks, while allowing savings in storage and bandwidth, do not provide, by themselves, support for document consistency enforcement. This problem has been specifically addressed in the context of another work [2], where a chunk-based system is extended with consistency enforcement applied at file level. On the other hand, associating consistency information with all chunks in the system would be very inefficient, and with other problems, since chunk content may be shared among several files.

In Semantic-Chunks, consistency is enforced at the semantic-chunk level, instead of at file level. Thus, consistency enforcement is performed by the Semantic-Chunk Manager. Semantic-chunks naturally suit themselves to the typical editing operations performed by users, i.e., centered in a fraction of the document sections, and inside

them, inserting, removing or editing some paragraphs, etc. This way, update and consistency information is kept on a semantic-chunk granularity. This is a ideal subdivision to provide increased concurrency in document edition, withstanding more updates and modifications while reducing update conflicts. This is achieved by avoiding, or at least, reducing false-sharing conflicts existing in other approaches (in systems where updates are regarded as a whole) arising from concurrent, yet unrelated, updates performed on documents.

Consistency meta-data, associated with semantic-chunks is encoded in an open, XML-based, flexible manner. In addition to use causality information, it can accommodate user voting schemes, authoritative updates, user leases, and custom hint messages. Semantic-chunks inherit, by default, and without overhead, consistency meta-data of their parent semantic-chunks.

Causality information annotating semantic-chunks is based on version-vectors (that may be compressed and subject to other optimizations [17, 21, 10]). Version vectors are applied hierarchically to semantic-chunks. Semantic-chunks are updated when its content data-chunks are modified (if it belongs to *Level-1*), or when there are insertions or deletions in its array of lower-level semantic-chunks. When the structure of a higher-level semantic-chunk remains unchanged, even in the presence of changes to lower-level chunks content, there is no need to transmit the higher-level semantic-chunk again, just the lower-level ones that were modified. This prevents having to transmit all the semantic-chunks in a document every time is subject to localized modifications.

Semantic-chunks mat be subject to user voting schemes. When users are confronted with (divergent) update conflicts, causal information cannot help. To solve this, users can insert, and retrieve, semantic annotations to the semantic-chunk stating their vote (possibly with associated weight) for a particular update candidate. Users vote based on the content they read and their opinion on it. This user-provided semantic information advises other users of which update to use and, if the number of replicas is immutable, may even allow automatic decision by the middleware. We stress that this voting is not based on an algorithm as in other epidemic propagation schemes, but on semantic information annotated by users.

Files and semantic-chunks may be subject to authoritative usage. This allows a specific user (owner or creator), or set of users (admins), to arbitrarily force their updates through other users with less privileges.

Semantic-chunks may be appended with semantic-data regarding lease of preference. It is a period of time indicated by an user, during which, he/she expects to produce another update to the semantic-chunk. This way, other users are advised that, if completed within time, this next update will have precedence to others w.r.t. conflicts.

Users can also annotate semantic-chunks with custom hint messages that will pop-up in the corresponding document region to inform other users. Several semantic annotations, possibly of different types, may be combined for any semantic-chunk. The middleware can adapt consistency granularity to suit file owner preferences, overriding default behavior.

An update is comprised of a set of semantic-chunks that have been modified. Updates are propagated in two ways, either i) implicitly, epidemically whenever two peers meet with neighboring devices, or ii) explicitly, whenever two or more peers meet and the file owner broadcasts a new update to explicitly overwrite all other replicas.

The ability to enforce consistency on a granularity larger than logging every singular operation performed, and smaller than *all-or-noting* complete file updates, while also exploiting additional semantic information defined by users, is key to provide high concurrency, low number of update conflicts, avoid centralized reconciliation, and ensure transparency w.r.t. *everyday* office applications.

3. IMPLEMENTATION ISSUES

The development of a preliminary Semantic-Chunks prototype is currently underway. It targets laptop machines, while taking portability to PDAs into account. It makes use of Bluetooth connectivity, .Net Framework and .Net Compact Framework (.Net CF), and Office applications. In this section we present its main design aspects, explain the issues involved, the limitations found, and the decisions taken.

Communication Services are developed in C#, and use native .Net (and .Net CF) support for web-service invocation. Requests received from other peers are answered by .Net Active Server Pages coded in C#, running on IIS or on a Mobile Web Server [15].

File System Extension is deployed as an IFS (Installable File System), coded in C++, to manage folder initialization, directory maintenance, navigation through subfolders, and file operations themselves. To leverage the use of managed code (usable both in .Net and .Net CF), most of the file system extension code is coded in C# and is invoked from the core C++ code via a documented *interop* hook [5, 20].

The main code of **Chunk-Database** and **Semantic-Chunk Manager** is developed in C#. This eases development w.r.t. C++, mainly because it is straightforward to integrate it with the Communication Services (that make use of the native support, in .Net and .Net CF, for web-service invocations). Semantic information annotating semantic-chunks, regarding consistency, is XML-coded and is also parsed and processed by C# code.

There is a reference count field associated with each data-chunk and each semantic-chunk. It is incremented/decremented whenever a semantic-chunk creates/drops a reference to a lower-level semantic-chunk or to a data-chunk. When it reaches zero, the semantic-chunk or data-chunk can be garbage collected by the middleware, since it is no longer part of any of the documents stored. This is performed lazily (i.e., when free memory reaches a low threshold).

Binary data, like data-chunks content, is sent over web services which imposes some penalties, and it is still an open issue [3]. Alternatives could be: i) using a lower-level communication protocol exclusively to send binary content, and use web-services for higher level communication, or ii) refine Semantic-Chunks Manager to generate (longer) XML-only descriptions of data-chunks content.

The **Application Enhancement Layer** is mainly coded in C#. It has some parts in VB.Net for clearer interaction with VBA-based Office automation, in order to extract and inject content, stubs, and semantic hints w.r.t. consistency. Document files and semantic-chunks are exchanged as XML representations of comprising semantic and data-chunks, much as a stripped-down version of OpenOffice [1] format. Except for the Pocket Outlook Managed API, Pocket Office still lacks substantial automation support. This is a feature long awaited due to the large number of applications and documents that use it. Nonetheless, convergence with the desktop versions has been moving forward (e.g., use of native Office file formats).

Finally, while data-chunk contents follow the document w.r.t. content and formatting, these are coded in an way independent of the actual Office file format that is proprietary, binary and rather opaque. Thus, when possible, it is easier to use automation to extract content than parsing/generating files. The latter is inadequate for a Semantic-Chunk system because the Office format has some idiosyncracies (e.g., saving a figure in a file, far from the position where it saves the text that surrounds it). To address this format incompatibility, when a file is copied to a folder that is not managed by Semantic-Chunks, it must be rendered by Office and saved with the regular application format. This may also be performed resorting to automation.

4. RELATED WORK

Semantic-Chunks is related to number of other projects regarding replication and consistency. Due to lack of space, we only address some. For a comprehensive survey, we refer to [18].

Bayou [21] is based on mobile-aware databases. Consistency is enforced by performing update operations in the same, well-defined order at all servers. This achieves eventual consistency among servers. Application-specific conflict resolution is performed by opaque dependency checks and merge procedures.

LBFS [14] was the first system to exploit file content similarities in order to save storage and bandwidth. It did not consider consistency, that was addressed in Haddock-FS [2], with file granularity.

Operations performed by applications are logged by IceCube [12, 19], in different clients, and later sent to a generical central reconciliation server. It heuristically performs sound re-scheduling of operations in order to minimize conflicts, based on constrains for each pair of operations.

The work presented in [7] aims to achieve substantial reductions in update transmission latencies, allowing sending updates with increasing level of fidelity, by leveraging information about document structure and content adaptation.

Replication of XML documents, in mobile environments, has been addressed in the context of Xmiddle [13].

With respect to other reflective middleware, sematic information regarding QoS non-functional aspects is extracted and defined declaratively by contracts in [4]. Compatible contracts can be combined straightforwardly. Conflicts among requirements from different contracts are solved based on priority. An hierarchical approach is also used in [6], in this case, to define channels in a publish/subscribe messaging model. Using semantic information in the context of transactions (another way of enforcing consistency), has been addressed in [11], to meet varying transactional requirements from applications.

5. CONCLUSION

In this paper we presented a novel approach to replication, concurrency, and consistency enforcement, with a proposed implementation. Semantic-chunks are a novel approach to cooperative document edition in ubiquitous environments. A *semantic-chunk* is a document region with relevance to applications and users, further annotated with semantic consistency information, in part provided by users. Since it is smaller than a file and semantically richer, it allows greater concurrency and better update merging with less aborts than current solutions. While providing more flexible consistency enforcement, Semantic-Chunks imposes a fraction of additional storage overhead w.r.t. LBFS (that does not enforce consistency) and HaddockFS.

Semantic-chunks do not impose modifications to applications, nor specific merge procedures, nor centralized conflict resolution. They are both intuitive and flexible w.r.t. users, and transparently manageable by the middleware.

We establish a middle-ground between two widely adopted families of approaches, *update-based* and *operational-based*. Thus, the problems of each one are avoided, while retaining the advantages of both. From the *operational-based* approach, we take increased concurrency, but without the need to modify applications. From the *update-based* approach, we take transparency w.r.t applications, but attempt at reducing update conflicts.

Being designed as a middleware layer, between the operating system and the applications, Semantic-Chunks architecture and implementation are developed following adaptive and reflective middleware techniques.

As future work, we intend to circumvent present Pocket Office's lack for automation support, by using content extraction/injection libraries as Application Adaptation Modules (e.g., parser/writer for Rich Text File, a format understood by Pocket Word).

We also intend to investigate the application of the Semantic-Chunks approach to another CSCW applications for content and document edition (e.g., an ad-hoc, ubiquitous Wiki system; OpenOffice applications with OOBasic enhancement layer).

We also wish to further develop Semantic-Chunks w.r.t. consistency guarantees, different consistency enforcement approaches, and providing update hints for conflict resolution.

We want to test the system with a typical set of users, once a complete user-friendly prototype is finished. Finally, we intend to measure improvements in concurrency and successful committed updates w.r.t. previous approaches.

6. REFERENCES

[1] Using openoffice.org's xml data format. http://books.evc-cit.info/book.php, OReilly & Associates, Inc., jul 2004.

[2] J. Barreto and P. Ferreira. A highly available replicated file system for resource-constrained windows ce .net devices. In *3rd International Conference on .NET Technologies*, 2005.

[3] A. Bosworth, D. Box, M. Gudgin, M. Nottingham, D. Orchard, and J. Schlimmer. Xml, soap and binary data. http://www.xml.com/pub/a/2003/02/26/binaryxml.html, feb 2003.

[4] R. Cerqueira, S. Ansaloni, O. Loques, and A. Sztajnberg. Deploying non-functional aspects by contract. In *The 2nd Int'l Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[5] M. Combs. Unmanaged to managed calls - call managed code from unmanaged code. http://www.codeproject.com/dotnet/bridge.asp, The Code Project, 2003.

[6] E. Curry, D. Chambers, and G. Lyons. Introducing reflective techniques to message hierarchies. In *The 2nd International Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[7] E. de Lara, R. Kumar, D. S. Wallach, and W. Zwaenepoel. Collaboration and multimedia authoring on mobile devices. In *Int'l Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Franciso (CA), USA, May 2003.

[8] P. Ferreira and L. Veiga. Mobile middleware: Seamless service access via resource replication. Technical report rt/08/2005 (extended version of the chapter included in Mobile Middleware, A. Corradi and P. Bellavista eds., CRC Press, 2005), INESC-ID Lisboa, april 2005.

[9] I. Greif, editor. *Computer-Supported Cooperative Work: A Book of Readings*. MORGAN KAUFFMAN, 1988.

[10] Y.-W. Huang and P. S. Yu. Lightweight version vectors for pervasive computing devices. In *International Conference on Parallel Processing Workshops (ICPPW'00)*, 2000.

[11] R. Karlsen and A.-B. Jakobsen. Transaction service management: An approach towards a reflective transaction service. In *The 2nd Int'l Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, 2003.

[12] A. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, Newport, RI, USA, August 2001.

[13] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. Xmiddle: A data-sharing middleware for mobile computing. *Wirel. Pers. Commun.*, 21(1):77–103, 2002.

[14] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, October 2001.

[15] N. Nicoloudis and D. Pratistha. .net compact framework mobile web server architecture. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/NETCFMA.asp, Monash University, Caulfield, Australia & MSDN, Microsoft, jul 2003.

[16] M. O. Rabin. Fingerprinting by random polynomial functions. Report tr-15-81, Center for Research in Computing Technology, Harvard University, Cambridge, MA, USA, june 1981.

[17] D. Ratner, P. Reiher, and G. Popek. Dynamic version vector maintenance. UCLA Technical Report CSD-970022, june 1997.

[18] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1), 2005.

[19] M. Shapiro, N. Preguiça, and J. O'Brien. Rufis: mobile data sharing using a generic constraint-oriented reconciler. In *IEEE International Conference on Mobile Data Management (MDM 2004)*, jan 2004.

[20] M. Struys. Asynchronous callbacks from native win32 code. http://msdn.microsoft.com/library/default.asp?url=/library/en-us /dnnetcomp/html/AsynchCallbacks.asp, PTS Software/MSDN, Dec 2003.

[21] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182. ACM Press, 1995.

[22] N. T. I. S. U.S. Department of Commerce/N.I.S.T. Fips 180-1. secure hash standard. Springfiled, VA, april 1995.