Data-Knowledge-Context: An Application Model for Collaborative Work

Lee Iverson
Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver BC, Canada
leei@ece.ubc.ca

ABSTRACT

For many years, researchers and software developers have been seeking to develop systems and applications to enable efficient and effective group work and organizational memory. The systems proposed and developed have in many respects had little impact on the effectiveness of group activities outside the laboratory. Other researchers have identified many of the challenges that groupware systems face, but these insights have done little to structure subsequent research.

We suggest that these difficulties are primarily due to an operating system model and a model of application development that has significantly restricted the ability of users to properly manage their own data and work products much less share them with others. Moreover, it is nearly impossible to effectively integrate collaborative activities with natural practices of work and communication. Instead, we propose an alternative system architecture, the DKC model, that places HCI, knowledge representation and management, and distributed, hypertext operating systems in a coordinated structure. We clearly delineate the roles of each of these aspects within the whole, collaborative environment. By adopting this model, we suggest that researchers and developers of both single-user and collaborative systems will be able to design effective, multi-platform, multi-application, and multi-workplace collaborative environments that may finally have some impact beyond the laboratory.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Patterns; D.4.3 [**File Systems Management**]: Distributed File Systems; H.2.4 [**Systems**]: Distributed Databases; H.5.4 [**Hypertext/Hypermedia**]: Architectures

Keywords

Computer Supported Cooperative Work (CSCW) (primary keyword) ; Database access / Information Retrieval ; World Wide Web and Hypermedia ; Software architecture and engineering

1. INTRODUCTION

Anyone who was there or has seen video of the 1968 "Mother of All Demos"[13] at which Douglas Engelbart first revealed combination of the mouse, hypertext, visual/interactive computing, multimedia workspaces, and shared environment, can't help but be impressed with both the vision and execution of technology in the service of collaborative work. Some who worked there have suggested that in terms of enabling flexible, integrated collaboration and organizational memory, few systems have ever risen to the level established in Engelbart's Augmentation Research Center [14] in the early '70s. In a certain respect, much of the research in groupware, hypertext and CSCW that has gone on since then is still trying to recapture the potential of that working environment. We would like to suggest that an understanding of certain aspects the evolution of these technologies and the relationships that these have with both social processes and, in particular, the patterns and models for development and delivery of software applications may explain the failure to be able to recapture that integration of technologies in support of collaborative work.

Much of the work in CSCW has focused on analysing the interfaces between technologies and group work activities in order to better facilitate effective, efficient collaborative processes and ensure that the technologies support, rather than disrupt, the social activities that this collaboration depends on. Among the most useful of these analyses as far as guiding the design of future systems are those that examine both the failures (e.g. office automation and videoconferencing) and successes (e.g. email and the Web) of the technologies designed to encourage and facilitate group work. In particular, we need to combine analyses of groupware technologies themselves[19], with an understanding of the social requirements for the success of those technolgies[1], and a deep consideration of the affordances of computer-mediated communication systems as far as their ability to enable collaborative activities[32].

Grudin, in an essentially pre-Web study of the legacy of the early work on groupware systems and their relative failure with respect to both enterprise-scale MIS and single-user productivity tools, identified eight "challenges" for develop-

ers[19]. We would like to highlight four of these, what he called "work vs. benefit", "critical mass", "unobtrusive accessibility" and "the adoption process". With the "work vs. benefit" challenge he suggested that one of the most critical failures was in creating systems and work environments in which the work required to make them function effectively is necessarily performed by those who get little benefit from the work they are doing. The "critical mass" issue has to do with the tendency to develop technologies or work practices that require full participation from the users to provide any benefit at all. An example would be group calendar applications, which often break down if even a single participant doesn't contribute to the collective scheduling and notification. He described as "unobtrusive accessibility" the problem of having the group coordination features of software applications being relatively infrequently used in comparison to the normal work practices. The consequence of this is that these collaborative functions need to be *more* transparent and usable than the typical task-oriented features of the collaborative applications. Finally, he highlighted the need for a clear and workable "adoption process" that takes into account such issues of heterogeneity of work environments and practices and the means by which users change their work habits and tools when necessary.

The usefulness of this analysis can be appreciated by considering the degree to which the success of the Web as a collaborative information sharing environment can be understood in comparison to much more capable hypertext environments that preceded it (e.g. Intermedia[33] or Microcosm[18]). I would like to suggest that, the combination of its simplicity, early standardization and open source implementation, contributed to an easy and seductive adoption storyline as well as a clearly unobtrusive accessibility. In terms of work vs. benefit, it accommodates a wide variety of different user types and goals from pure browsers to large-scale communities of interest, to work-focused intranets, to small-scale vanity publishing. It's simplicity and flexibility has allowed it to adapt to a wide variety of different social environments and easily develop a critical mass of users well beyond it's initial conception as a tool for efficiently sharing scientific information. In fact, this explosive growth beyond the initial conception and plan even harks back to another of Grudin's observations, that there is a "failure of intuition" for developers in designing groupware systems in being able to predict the impact of these systems on their intended user communities.

Ackerman[1] built on Grudin's work and expanded it by considering the social dynamics of cooperative work more deeply and thus emphasized a number of different aspects of the problem, especially with respect to particular issues that we don't yet know how to handle from a technological point of view. He pointed out that social processes are fluid and nuanced and that systems for sharing information and opinion need to reflect that and provide simple ways of managing the subtleties and dynamics of our trust in those we are or might be collaborating with. He pointed out the importance of synchronous collaboration, especially in terms of the "presence" and "visibility" of our collaborators and their work. Finally, echoing and expanding on Grudin's assertion that intuition on the effects of design choices for group work, he argued strongly for a co-evolutionary approach to the development of these systems, in which there is a strong assumption that the technology will affect its user communities and that these effects should be studied, understood and force the adaptation of the software and other tools to the needs of the communities.

Finally, Whittaker[32] examines the the existing literature on the affordances of various computer-mediated communication (CMC) media and compares these with the needs of collaborative workers in terms of the avoidance of redundancy, the exploitation of varying expertise and experience within groups. He highlights the need to support both synchronous and asynchronous modes of communication and to explore shared experiences through shared environments much more deeply. When combined with some of the recent emphasis on "common ground"[8] or mutual knowledge development using CMC technologies[10], there is a clear indication that for cognitive tasks that depend critically on efficient communication of knowledge and the the recognition and resolution of conflicts, synchronous and asynchronous communication pathways and recordings of conversations and conclusions (e.g. email archives) all contribute to collaborative work in complementary ways.

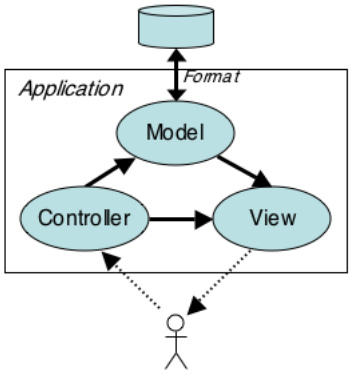
With all of this knowledge then, why are we still not able to develop effective collaboration environments for our user communities? We would like to suggest that one of the primary stumbling blocks is a fundamentally uncooperative model for the development and dissemination of software applications. Given the background knowledge outlined above, we will show how this model inhibits effective collaboration, and then propose an alternative that may will allow for the development of much more effective collaborative tools without requiring users to abandon their entire existing software environment and work practices.

2. CURRENT APPLICATION MODELS

In many ways, the model-view-controller pattern[21] characterizes the modern accepted best-practice approach to the development of effective user-oriented software applications (see Fig. 1). It describes a modularization of functionality in interactive software applications in which the user interacts directly with a controller, that in turn interprets the user's actions to modify both the application's internal data model and the representation of that model on the computer screen (the view). In order to maintain a persistent record of the model (and view), the application designer either adopts or designs a data format which encodes the model and stores it in a file on some filesystem (networked or not). Notably, since it is often difficult to relate these internal models of tasks and user activities in applications to standardized data formats it is normal practice to create a new format specific to every significant new application or task-oriented system.

One of the consequences of this design pattern is that it leads to an expectation that persistent data will be encapsulated in these, often complicated or deliberately obscure, external data formats. Moreover, it is assumed that a new data format is required for each new application or data model. When combined with economic incentives within the software industry for consumer lock-in and mandatory upgrade paths, it becomes clear that there is a huge opportunity for a user's data and knowledge to be "captured" and even

Figure 1: Model-View-Controller pattern with external file format application model. Note that the persistence of the model (and possibly view) is instantiated by the definition of a document format and by the process of encoding and decoding between the model and this external format, which is stored via a filesystem.



held hostage by the applications that create and manipulate them. We refer to this as the "tyranny of format". For software vendors, the choice to keep a data format proprietary is seen as one of the best tools for ensuring customer dependence. Since access to the user's own accumulated work and knowledge is necessarily mediated by the application, the user can only access their own intellectual output by using the vendor's applications. If this lock-in is effective, then it enables other potential manipulations such as subtly or radically changing data formats with upgrades to applications. The final consequence of this practice comes when it is aligned with a move to make more applications "leased", such that they require the maintenance of an active license in order to continue to function. In these cases, failure to subscribe and pay an annual fee can mean that a user would be prevented from even continuing to access their own data,

Another consequence of this application model derives from the observation (related to the second point above) is that collaborative activities are also "captured" by the data formats and depend on the goals of the vendor in supporting collaboration within the application and not on the needs of different user communities for collaborative facilities. In essence, since the data model and task-oriented knowledge are inaccessible outside the application (except via external data formats or application-provided communication paths) there is very little possibility of developing general personal information management tools except at the granularity of "files" or documents stored in filesystems. Important classes of information management and collaboration tools that *could* potentially be implemented relatively independently of task-oriented applications are impossible to implement without direct assistance from these data-capturing applications (e.g. personal information management, hypertext linking, anchored conversations[7], etc.). Moreover, the potential for reuse of information inside captured documents by other applications is limited by the application developer's or vendor's commitment to support such reuse.

One response to this problem (whether the problem was stated in these terms or not) is to create "standard" data formats and even languages (such as SGML or XML) for expressing a variety of data formats in standard form. This is admirable and necessary, but probably not sufficient. It's success depends critically on patterns of adoption by vendors and developers and the need to satisfy many different parties in the standardization processes often means that the standards developed are extremely complicated and generally unmanageable by any but the largest corporations or communities of developers.

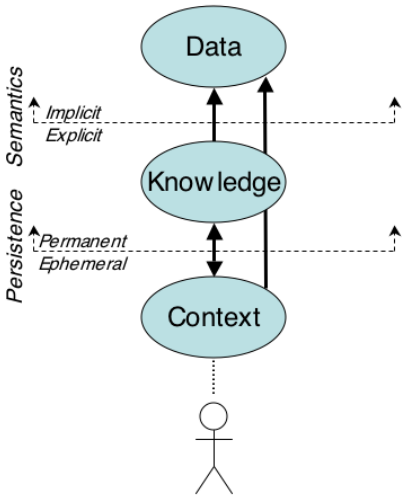
A case in point is the widely-perceived complexity and unusability of the current XML Schema: Part 1 (Complex Datatypes) standard[31]. This has been characterized as a conflict between the data-oriented (i.e. database management) and text-oriented (i.e. SGML and HTML) communities and the difficulty of resolving their often competing requirements[9]. We would like to suggest that in addition to these difficulties is the fundamental confusion embedded in the XML standards between data models and syntax. XML, as flexible and adaptable as it may be, is still a markup language, a syntax designed primarily for "marking-up" primarily textual documents. It has been adapted with mixed success to the goal of representing and exchanging fundamentally non-textual data resources (e.g. XML-RPC and SOAP), but it retains its roots as a language for expressing text. We suggest that a step beyond XML would clearly separate data model from the (possibly numerous) languages for expressing that data model and build a collaborative foundation around that. The DKC model we are proposing begins with exactly that observation.

3. THE DKC MODEL

The Data-Knowledge-Context (DKC) model (see Fig. 2) is a layered application development model and system architecture that clearly separates concerns between a general operating system layer (the *Data* layer) and two semantically rich layers above it (the *Knowledge* and *Context* layers). It separates these concerns based on persistent vs. ephemeral storage and explicit vs. implicit semantics. The Data layer is a generic data storage infrastructure designed around an extensible data modelling language, communication and data distribution facilities, both structural and content-based search, change auditing and a security and privacy system designed to promote patterns of data sharing that match the nuances of trust patterns in collective work. It needs to be platform-independent, language and programming language-independent and application-independent. The Knowledge layer is a persistent store of explicitly semantic knowledge intended to provide the facilities for storing and accessing semantically rich depictions and interpretations of the world. The Context layer is the ephemeral, task and user-oriented interface to this world of data and knowledge.

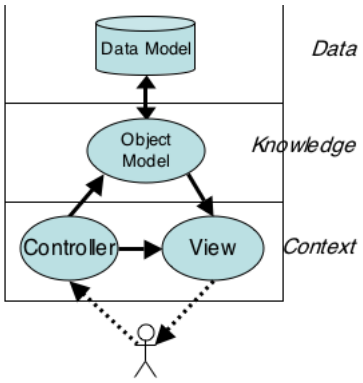
In terms of subverting the dominant MVC application paradigm (see Fig. 3, the Data layer stores the model as a structured data model and provides all of the necessary facilities to manage this data model by multiple users, with multiple data formats and with multiple patterns of access and communication of the components in this model. The view and controller parts of the application are aspects of the Context layer, designed to manage task-based interactions and visualizations of the data for the user and task being targeted.

Figure 2: The DKC Model, a three-layer model of Data, Knowledge and Context layers. The Context layer is equivalent to the client application.



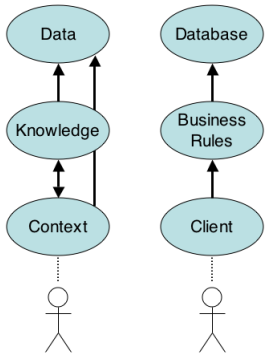
The Knowledge layer should capture, and make independently available to other Contexts, the business rules and logic that can be abstracted out of the application. Moreover, it will be able to store and manage a wide variety of personal and group information management facilities that should make semantic search and reuse of data more transparent. However, it should be noted that the Knowledge layer is an optional part of these systems. In general, the Knowledge layer should be seen as an opportunity for developers and users to develop new facilities for managing and exploiting knowledge that is now available outside of the data-capturing applications.

Figure 3: MVC mapping to DKC layered model. The View and Controller modules are in the Context layer and the Model may be an object model, knowledge model or simply a data model. The application is essentially just the View and Controller modules.



In many ways, this model can be compared favourably to the database-oriented two-tier and three-tier models (see Fig. 4) for enterprise level applications. The Data layer should be, in fact, expressed in terms of database-like language and the Knowledge layer can be seen to be the expression of the business rules for managing that data. The Context layer is then clearly associated with the user-interface issues at the top of the two-tier or three-tier layers. We do not claim that the DKC model is in any way independent of these architectures. In fact, it can be seen as a way of taking the lessons learned from those models and applying them to the next generation of flexible, deeply user- and task-oriented applications for a much wider class of users than traditionally supported by the enterprise-level systems. We do this, specifically, by describing requirements and concerns that the layers must or may support that go beyond the traditional concerns of the tiers in these models and focus explicitly on allowing collaborative work between loosely-connected individuals and groups without the boundaries imposed by data-capturing application models. Moreover, we make explicit the need for systems built on these foundations to be able to "play well with others" and interact effectively with entrenched or idiosyncratic applications and work practices.

Figure 4: DKC and the three-tier enterprise model. The only major difference here is that the Knowledge layer does not isolate the database from the client/context layer.



One other class of applications that we are aware of already more or less ascribes to this model, shared virtual worlds[24]. In these environments, the world model is, in fact, separated from the front-end application and, whether distributed or centralized, is managed via a series of transactions between the client and server(s). Moreover, the communication between users of these systems is managed (and sometimes recorded) by the central server(s). Thus, these servers are implementing the Data layer in this kind of system, including both persistent storage and facilitated communication. They do not, however, tend to make the business rules of the system available to their users or client applications in a way that could be exploited with automatic reasoning tools. In that sense, they implement the Data and Context layers, but the Data layer is not entirely pure, being a combination between the Data and Knowledge layers. Moreover, they tend to not support the mixture of asynchronous and synchronous access that may be necessary for supporting collaborative work practices. That said, in terms of implementation strategies and techniques they provide an excellent example of the kinds of distributed technologies and design patterns that might go into a DKC-based system.

So far, we have only glossed over the concerns and respon-

sibilities of these three layers with reference to metaphors and existing systems. In order to make this model applicable to design and analysis, we must make the concerns and requirements of the component layers more explicit. Appropriately, we will start with the Context layer, the one closest to the users and tasks to be supported.

3.1 Context Layer

The Context layer is the primary location of HCI-related concerns. In a pure implementation of the DKC model, in fact, the Context layer *is* the client application itself. As such, it manages all interactions between the user and the data models that are being manipulated. It encompasses user-interfaces including visualization, presentation, and interaction. For purposes of the trust management that must be communicated to lower layers, it is also where authentication maintenance must be managed, ensuring that the user claimed to be interacting with the models is in fact the person that the permissions have been entrusted to. In the future, as these technologies mature, it is within this layer that we will see the deployment of user and task-modelling and more sophisticated knowledge capture (e.g. speech recognition and transcription). For more significantly hardware-dominated collaborative application environments (such as instrumented spaces or augmented environments), it is likely that the initial integration of the mixed media will have to happen at this layer in order for context to be recognized and captured.

It is named the Context layer primarily because although context is largely recognized as important in representing and capturing knowledge it is generally not considered explicitly within the modelling of systems (although this usage is related to "context-aware computing", the breadth of our use of the term is obviously broader and come from discussions with Charlie Ortiz at SRI). In our estimation, user-interfaces, user- and task-modelling and knowledge capture are all driven by ephemeral semantic context and we emphasize this with the name.

3.2 Knowledge Layer

The Knowledge layer is responsible for the representation and management of explicit, persistent semantic knowledge. The goal of identifying this as a separate layer in a system architecture is to focus on the reusability of such knowledge and services outside the boundaries of particular contexts and application-based constraints. This is the traditional domain of knowledge representation and management systems, models and technologies and we hope to provide a means by which these may be made available to all knowledge workers.

Significantly, we explicitly recognize the need for a variety of different semantic services (see Fig. 5), ranging from simple representational systems (e.g. ontological models) through full-fledged reasoning systems (e.g. traditional knowledge bases).

3.2.0.1 Declarative Semantics

In that respect, it should be clear that the storage and management of Semantic Web[4, 16, 11] data and the services necessary to take advantage of this information would lie in this layer. Some work has recently been done on using ontological models as foundations for higher-levels of information integration. In general, whether the models are being produced automatically, semi-automatically or completely under user control, there is a significant advantage to be gained by assuming that their expression and use is made distinct from both the Data layer and the Context layer. Whether these semantic models are considered to be data or metadata, a clear constraint that comes from the current representations being used to codify this declarative knowledge is that they represent information *about* other objects, and thus rely inherently on models of reference to objects. Any object or granularity of object that cannot be referenced in a Data layer, cannot be the subject or object of a declarative semantic statement. Providing a more universal means of referencing data at a variety of granularities should be one of the main constraints on the design of the Data layer.

3.2.1 Procedural Semantics

More interesting, we claim, is the observation that procedural knowledge is just as important and potentially reusable as the kinds of declarative knowledge traditionally associated with knowledge representation. What do we mean by procedural knowledge? Well, object models for one. When data is tied to behaviours to form behaving objects, there is a clear sense in which those objects explicitly represent a semantics for the data they encapsulate. We suggest that in many cases it will enhance the overall patterns of reuse of data and semantics if we explicitly separate these procedural semantics of objects from the implicit semantics of structured data. A claim that we rely on, and we hope to address it in future work, is that we believe that object models limit reuse of information in a way that structured and semi-structured data models (at the Data layer) do not. Given that contention, it makes sense to separate these concerns and allow reuse and repurposing of data separately from the procedural constraints imposed by an object model. In many cases, of course, we may need to limit this independence (e.g. in financial systems where there are real hard constraints on the data model that cannot be represented in the data model constraint language), in which case we may need to restrict modification of the data model by using the knowledge layer in the same way as the business rules layer in the three-tier enterprise model, to isolate the client from the actual data model. In general, however, this an optional usage in the DKC model and needs to be supported but not required.

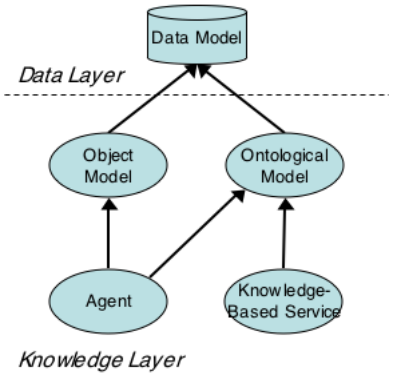
This is not an entirely radical idea if we consider the experience of agent-oriented systems in which the procedural and declarative semantics of agents are necessarily combined into a single package so that agent brokers may reason about them in order to coordinate services5. In these systems, agent languages[23] such as KQML[17] are combined with mobile and distributed objects to produce mobile, self-describing active agents. The difficulty is often ensuring that the declarative and procedural semantics stay harmonized as the agent's implementation evolves.

3.2.2 Knowledge Services

In addition the the basic declarative semantics, it is necessary in many environments to provide automatic reasoning

services that can draw conclusions or generalizations from the semantic models represented in a declarative fashion. These KBS-style systems, we argue, are appropriately situated at this knowledge layer and should be independent of the view-control elements of the context layer above it. In that respect too, we would classify web services (in so far as they are semantically well-defined) and data mining services as aspects of the DKC Knowledge layer.

Figure 5: Relationship between Data Model, Object Model and Knowledge Model within the Knowledge layer. Agent services typically combine Object and Knowledge models while Knowledge-Based services provide logical inference services on the Knowledge Models.



How are the Context and Knowledge layers related? The Context layer is responsible for capturing contextual information and formulating it in such a way that it can be coded and reused by the Knowledge layer. The Knowledge layer, in the mean time will be typically responding to semantically-rich, context-dependent queries from the Context layer and and providing services to enhance the Context layer's interfaces. Meanwhile, the Data layer sits underneath both of these and provides the universal, operating-system like services that are required to implement both.

3.3 Data Layer

Since the Data layer sits underneath the two other layers, it is responsible for all aspects of the system implementation that are not rightly associated with either. In particular, it is a layer for persistent storage but not explicit semantics. We suggest that to truly approach the needs expressed by Grudin[19], Ackerman[1] and in this paper, it needs to support the following set of mixed functionalities.

Data Model The observations from experience with enterprise-level architectures, shared virtual worlds and even modern dynamic web sites suggests that the best foundation for sharing and flexible reuse of data is a general and flexible data model. Moreover, with our abstraction of the DKC model out of the model-view-controller architecture, we can assume that the models that need to be made persistent are structured and constrained.

Database View Given the emphasis on a structured, constrained data model, it should be clear that most of the basic interactions with the model must be managed exactly as they are in a modern, multi-user transactional database. In fact, we suggest that implementing a data layer on top of either an object database or relational database is the best option for bootstrapping such a system. One point though is that since we are relating the model directly to the development of user-level applications, we probably cannot support the kind of conceptual mismatch between programming language data models and relational models. Instead, it seems essential that the data layer appear in this respect as a distributed object database.

Filesystem View We have pointed out clearly above that it is not a luxury or afterthought to support existing applications, work patterns and data formats. As such, it will be essential to maintain a connection between the database view and a distributed filesystem view. For this reason, we suggest that a Data layer must have some extensible architecture for relating data models and formats and that the availability of these crosswalks be made explicit and public. Moreover, the Data layer should be able to incorporate and provide facilities for accessing remote filesystem-like resources (e.g. NFS, HTTP, IMAP, etc.) with the same APIs and modeling tools available on the database side.

Asynchronous and Synchronous Interaction Since effective work patterns and communication for collaborative activities involves both synchronous and asynchronous communication, and end-user devices may move on and off the network, it will be necessary to provide means of interacting with both the database and filesystem layers in both synchronous and asynchronous modes. This would potentially mean that when connected, a user could be interacting with an interactive, shared data model in an effectively shared environment with any other connected users of a data repository. When offline, the user should still be able to work, but with an interaction mode that is queued and then synchronized when eventually reconnected.

Granular Reference We argued that granular reuse of data was important. Moreover, the key enabling technologies for Semantic Web services rely on the ability to reference content within files. If we hope to bring knowledge management and hypertext capabilities to all data formats (as we stated above) then it is clear that we need to provide some sort of universal, granular referenceability of content *inside* arbitrary files.

Change Auditing and Synchronization In order to enable the orderly transition from disconnected to connected use and resynchronization of resources, it is necessary to have some kind of change tracking and then synchronization methods. We suggest that the flexible data model and granularity of reference can provide a mechanism that can allow for universal change tracking and synchronization for any kind of document format via identification of data models and independent history auditing for all data model objects within documents.

Flexible Security and Privacy Clearly providing a means for users to flexibly manage the trust environment for

allowing others to reuse and even manipulate their data resources is essential for effective sharing and collaboration. Moreover, we suggest that the appropriate granularity for this security management is at the level of the sub-document objects within the data model.

Search for Content and Structure Clearly for purposes of data mining and information management, providing a universal search infrastructure is essential. Moreover, we suggest that this searchability be presented at both content and structural levels.

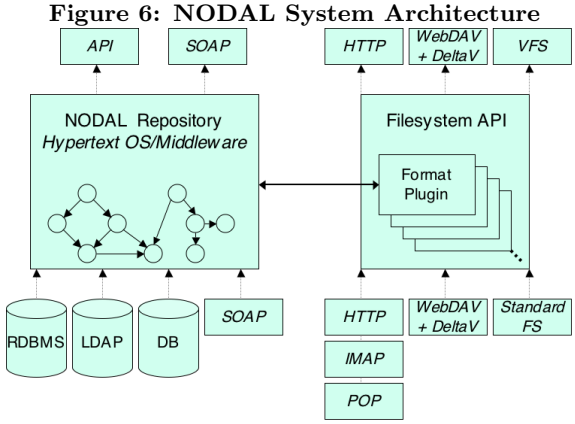
So, with these requirements, we have a recipe for the Data layer that could effectively support the Knowledge and Context layers above it while enabling application-independent collaboration and knowledge services. This Data layer would have to appear as both an active object database with message-oriented middleware facilities and as a distributed, synchronizing filesystem (such as in software change management (SCM) systems such as CVS[3], Subversion[15] or BitKeeper[5]). From a filesystem point of view, potential existing candidates for this aspect of the system would be the distributed filesystems such as Coda[30], Oceanstore[22], and ADHocFS[6].

These systems do not, however provide rich document modeling facilities that would allow the to connect the filesystem view with the database view. There are however, existing systems that either already implement this kind of facility or are moving in that direction, such as libferris[25], the BeOS filesystem[2], and the the WinFS filesystem that is positioned as part of Microsoft's Longhorn OS initiative[26]. In particular, the BeOS and WinFS systems seem to be the best systems to begin to use to begin to evaluate the potential for the DKC model as a new end-user application model. Unfortunately, while certainly integrating the database/filesystem model in order to provide "the best of both worlds" to their users, both systems concentrate on using the database elements in a very traditional way and in terms of the filesystem aspects, simply use it to replace the "directory" aspects of the filesystem model. This is quite similar to the Presto[12] system developed at Xerox PARC as a backend for a new kind of desktop and filesystem metaphor that de-emphasized hierarchy and names of digital objects in favour of rich, semantic properties and an integrated search facility (a clear precursor of the WinFS system). An open-source project that appears to have great potential as such a foundation is the GNOME Storage[27] project. It is clearly motivated by many of the same concerns and perceived needs as the analysis above, but has, at this point a definite platform-specific flavour that may significantly limit its impact. Unfortunately, in each of these systems the documents themselves are still represented as bitstreams to be interpreted by the consuming applications. Thus, where they may greatly enhance opportunities for new styles of information organization, they leave in place the reuse and collaboration limitations that result from the "tyranny of format". Moreover, they are each tightly integrated with the underlying operating systems, which clearly limits their reusability. Thus none of these systems seem to really meet the full requirements of the Data layer described above.

There are however, a few of systems described in the literature or in active development that seem to have some chance of meeting the full set of requirements. HOSS[29], a hypermedia operating system appears to have been built around many of the same principles outlined above but with not as much emphasis on extensibility and backwards compatibility with existing formats and applications. It did, however, seed the development of a new approach to managing data and hypertext called *structural computing*[28] that emphasizes structure over data. It would seem that the Data layer described above would qualify as a structural computing environment. We plan to explore the alignment of these two paradigms in the future.

Since none of these systems meet our full set of requirements, we are pursuing a project that directly addresses the goals and requirements for such a Data layer. We refer to it as NODAL[20], the Network-Oriented Document Abstraction Library (see Fig. 6). As can be seen from the architecture diagram, its core features are the Hypertext OS/middleware, a database-like layer and its connection with a standard filesystem API. The model is extensible with plugins for different storage substrates (RDBMS, LDAP, etc.), document formats, referenceable filesystems (e.g. HTTP, IMAP, POP, local FS) and filesystem views (e.g. HTTP, WebDAV+DeltaV, NFS). It is designed around a document-oriented data modelling language that allows data models to be associated with a plugin-extensible set of file formats (either on the repository or client side). The data model is built around a set of Node types that are strongly-typed collections of properties organized as Records, Sequences or Maps. The Record types correspond to Pascal-like record types or database tables, while the Maps represent general dictionary-like structures and the Sequences are list-like ordered sequences. These structuring elements are each individually referenceable by URI and hold potentially complete metadata describing change history, security constraints and attribution.

Figure 6: NODAL System Architecture



4. CONCLUSION

Having summarized existing work to identify some of the requirements to create systems to enable effective collaborative work, we identified another endemic technological barrier to these collaborations that we called "the tyranny of format". The emphasis on data formats as the units of collaboration in most modern user-centered applications, we

argue, severely constrains the ability of users to engage in the kinds of collaborative activities they require and "captures" their accumulated work in applications that they may have little control over. We identify this format tyranny with the model-view-controller pattern used in the development of modern interactive software applications. In response to this, we introduced the Data-Knowledge-Context (DKC) model for collaborative applications that combines some of the modularization of the MVC model with the distributed application view of the three-tier enterprise application model and shared virtual environments.

In analyzing the DKC model and describing the relationships between the three layers, we were able to derive a clear and comprehensive set of requirements for the Data layer that would form the universal substrate for such a system development model. This new view of an operating system/middleware layer underneath collaborative applications resonates with a number of systems that have been developed in the past and which are now in development. We hope that with the description of this model, we will be able to seed the movement away from the existing models and towards more effective systems for human-centered collaboration. We have even described a means by which we can do this without sacrificing existing users and applications.

5. REFERENCES

[1] M. Ackerman. The intellectual challenge of cscw: The gap between social requirements and technical feasibility. *Human-Computer Interaction*, 15:179–203, 2000.

[2] Be Inc. *The BeBook*. 1997.

[3] B. Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[4] T. Berners-Lee. The semantic web roadmap. `http://www.w3.org/DesignIssues/Semantic.html`, 1998.

[5] BitMover Inc. Bitkeeper pro. `http://www.bitkeeper.com/Products.BK_Pro.html`.

[6] M. Boulkenafed and V. Issarny. Adhocfs: Sharing files in wlans. In *Proceeding of the 2nd IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA, April 2003.

[7] E. F. Churchill, J. Trevor, S. Bly, L. Nelson, and D. Cubranic. Anchored Conversations: chatting in the context of a document. In *Proceedings of the CHI 2000 Conference on Human Factors in Computing Systems*, pages 454–461, The Hague, The Netherlands, 2000. ACM Press.

[8] H. H. Clark and C. R. Marshall. *Definite reference and mutual knowledge*. Cambridge University Press, 1981.

[9] M. Classen. Schema wars: Xml schema vs. relax ng. `http://www.webreference.com/xml/column59/`, 2002.

[10] C. D. Cramton. The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3):346–371, 2001.

[11] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.

[12] P. Dourish, K. Edwards, A. LaMarca, and M. Salisbury. Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction*, 6(2):133–161, 1999.

[13] D. C. Engelbart. The mother of all demos. `http://sloan.stanford.edu/mousesite/1968Demo.html`, 1968.

[14] D. C. Engelbart and W. English. A research center for augmenting human intellect. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 395–410, 1968.

[15] B. B. et. al. Subversion handbook. `http://subversion.tigris.org/files/documents/15/576/svn-han`

[16] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Oil: An ontology infrastructure for the semantic web, 2001.

[17] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[18] A. M. Fountain, W. Hall, I. Heath, and H. Davis. MICROCOSM: An open model for hypermedia with dynamic linking. In *European Conference on Hypertext*, pages 298–311, 1990.

[19] J. Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):92–105, 1994.

[20] L. Iverson. NODAL: A filesystem for ubiquitous collaboration. `http://nodal.sf.net/NODAL-WhitePaper.html`, 2001.

[21] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August/September 1988.

[22] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[23] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages. *Intelligent Systems*, 14(2), 1999.

[24] R. Lea, Y. Honda, K. Matsuda, O. Hagsand, and M. Stenius. Issues in the design of a scalable shared virtual environment for the internet. In *Proceedings of HICSS'97*, January 1997.

[25] B. Martin. About libferris. `http://witme.sourceforge.net/libferris.web/`.

[26] Microsoft Corp. Welcome to WinFS. `http://longhorn.msdn.microsoft.com/lhsdk/winfs/daovrwelcometowinfs.aspx`.

[27] S. Nickell. Gnome storage. `http://www.gnome.org/ seth/storage/`.

[28] P. Nurnberg, J. Leggett, and E. Schneider. As we should have thought. In *Proceedings of the Eighth ACM Conference on Hypertext*, pages 96–101, Southampton UK, 1997.

[29] P. J. Nrnberg, J. J. Leggett, E. R. Schneider, and J. L. Schnase. Hypermedia operating systems: A new paradigm for computing. In *Proceedings of Hypertext '96*, pages 194–202, 1996.

[30] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[31] The World-Wide Web Consortium. Xml schema: Part 1 (structures). `http://www.w3.org/TR/xmlschema-1/`, 2001.

[32] S. Whittaker. *Theories and Methods in Mediated Communication*. MIT Press, 2003.

[33] N. Yankelovich, J. B. Haan, N. Meyrowitz, and S. Drucker. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21(1):81–96, 1988.