

# 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. The systems developed have often had little impact on the effectiveness of group activities outside the laboratory. We suggest that these difficulties are primarily due to a model of application development that has significantly constrained users in properly managing their work products much less share them with others.*

*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.*

## 1. Introduction

Much work in CSCW has focused on analysing the interfaces between technologies and group work activities in order to facilitate effective 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 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[9], with an understanding of the so-

cial requirements for the success of those technologies[1], and a deep consideration of the affordances of computer-mediated communication systems in enabling collaborative activities[17].

Grudin, in a study of the failures of early groupware systems, identified eight "challenges" for developers[9]. We would like to highlight four of these: "work vs. benefit", "critical mass", "unobtrusive accessibility" and "the adoption process". The *work vs. benefit* failure happens when we create systems in which the work required to make them function effectively is necessarily performed by those who get little benefit from the effort. The *critical mass* issue has to do with technologies or work practices that require full participation from all users to provide any benefit at all (e.g. group calendar systems often break down if even a single participant doesn't contribute). Since the group coordination features of software applications are relatively infrequently used in comparison to other features, he suggests that these collaborative functions need to demonstrate *unobtrusive accessibility* and be *more* transparent and usable than the task-oriented features. 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.

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. 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 trust in those we are 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, Whittaker[17] examined the literature on the affordances of various computer-mediated communication

(CMC) media and compared these with the needs of collaborative workers. He highlighted 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 work on common ground[4] or mutual knowledge development using CMC technologies[6], there is a clear indication that for cognitive tasks that depend critically on efficient communication of knowledge and 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? 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.

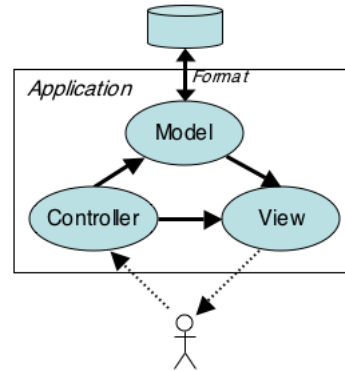
## 2. Current Application Models

In many ways, the model-view-controller pattern[11] represents the modern accepted best-practice for the development of effective user-interfaces (see Fig. 1). It describes a modularization of functionality in interactive applications in which the user interacts directly with a controller, that in turn interprets the user's actions to modify both an 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. Notably, since it is often difficult to relate these models to standardized data formats it is normal practice to create a new format specific to every significant new application.

One of the consequences of this design pattern is that it leads to an expectation that persistent data will be encapsulated in these external data formats. When combined with economic incentives within the software industry for consumer lock-in, it becomes clear that there is an opportunity for a user's data and knowledge to be "captured" and even 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 one of the best tools for ensuring customer dependence (and thus "loyalty"). Since access to the data is necessarily mediated by the application, the user can only access their own intellectual output by using the vendor's applications.

Another consequence of this application model is that

**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.**



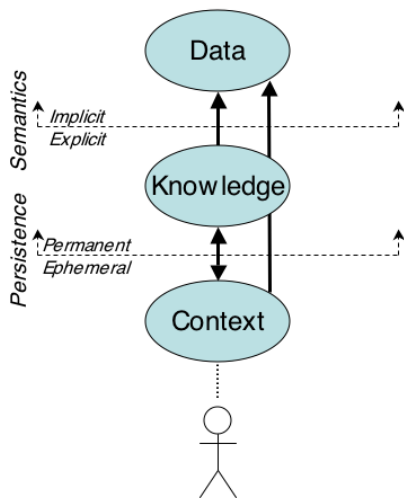
collaborative activities are also "captured" by the data formats and applications. The collaboration facilities available thus often depend more on the goals of the vendor than on the needs of the users. Since the data is thus inaccessible outside the application there is very little possibility of developing general personal information management tools except at the granularity of "files" or documents stored in filesystems (e.g. document management systems). 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, annotation, anchored conversations[3], 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 is to create "standard" data formats and languages (such as SGML or XML) for expressing a variety of data formats. This is admirable and necessary, but probably not sufficient. It's success depends critically on buy-in from vendors and developers. Moreover the need to satisfy many different parties in a standardization process often means that the standards developed are complicated and unmanageable.

A case in point is the widely-perceived complexity and unusability of the current XML Schema: Part 1 (Complex Datatypes) standard[16]. This has been characterized as a conflict between the requirements of the data-oriented (i.e.

database management) and text-oriented (i.e. SGML and HTML) communities [5]. More fundamental is the confusion embedded in the XML standards between data models and syntax. XML is a markup language, designed primarily for "marking-up" textual documents. It has been adapted 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.

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

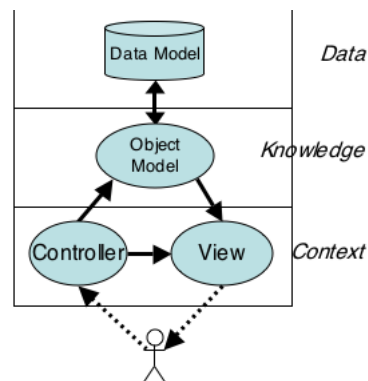


### 3. The DKC Model

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, 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. 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.

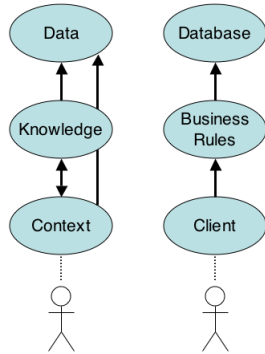
**Figure 3. MVC mapping to DKC layered model. The View and Controller modules are in the Context layer and the Model may be an object model, knowledge model or simply a data model. The application is essentially just the View and Controller modules.**



Whereas the MVC application paradigm (see Fig. 3 assumes little about the persistence of the model layer, the DKC Data layer stores the model as structured data and provides all of the necessary facilities to manage this data model by multiple users with multiple patterns of access and communication. 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. 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.

In many ways, this model is derived from the two-tier and three-tier models (see Fig. 4) for enterprise level applications. The Data layer can be implemented by a database and the Knowledge layer can be interpreted as the business rules for managing that data. The Context layer then contains the user-interfaces at the top of the two-tier or

**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.**



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.

So far, we have only glossed over the concerns and responsibilities of these three layers with reference to metaphors and existing systems. In order to make this model useful for design and analysis, we must make the concerns and requirements of the 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 user and task 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 and thus contains the user-interfaces including visualization, presentation, and interaction. In the future, as these technologies mature, it is within this layer that we will see the deploy-

ment of user and task-modelling and more sophisticated knowledge capture (e.g. speech recognition and transcription).

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. 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, management and exploitation of explicit, persistent semantics. We make this a separate layer in the system architecture to enable the reuse 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 and we hope to provide a means by which these may be made available to all knowledge workers within application contexts.

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).

**Declarative Semantics** The storage and management of Semantic Web[2, 7] data and services would lie in this layer. 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 data and context. This leads to one of the most important constraints on the Data layer, the need for a universal system of reference. Since semantic web technologies are built on URI-based reference, anything that cannot be referred to in a URI cannot be the subject or object of an RDF or OWL statement. Thus, providing a universal means of referencing data at a variety of granularities should be one of the main requirements on the design of the Data layer.

**Procedural Semantics** The separation of data and semantics has another consequence though, the separation of method from structure. In essence, we argue that for maximum generality and reusability, the procedural knowledge embedded in program functionality or object methods be separated from the data structures manipulated by these procedures. 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 represent the semantics of the data they encapsulate. We suggest that in many cases it will enhance the reusability of both data and semantics if we explicitly separate these procedural semantics of objects from the implicit semantics of structured data. We claim (and hope to prove in future work) that object models limit reuse of information in a way that structured and semi-structured data models do not. If this is true, then it makes sense to allow reuse of data separately from the procedural constraints imposed by an object model. The Knowledge layer is thus not usually an isolation layer (as it is in the enterprise model). In many cases, however, we may need to limit this independence (e.g. in financial systems where there are hard constraints on the data that cannot be represented in the data model constraint language) and filter all data updates (but not necessarily access) through the Knowledge layer. In general, however, this is an optional usage in the DKC model and needs to be supported but not required.

A hybrid of these two semantic models is the agent model, in which the procedural and declarative semantics are necessarily combined so that agent brokers may reason about them in order to coordinate services<sup>5</sup>. In these systems, agent languages[13] are combined with mobile and distributed objects to produce mobile, self-describing active agents.

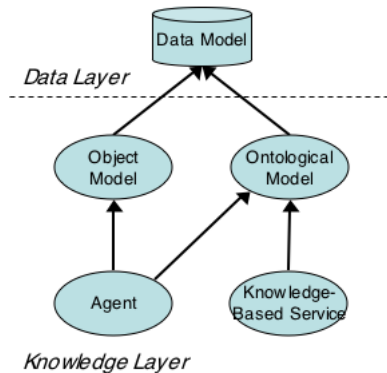
**Knowledge Services** In addition it is necessary in many environments to provide automatic reasoning services that can draw conclusions or generalizations from the semantic models represented in a declarative fashion. These KBS-style systems, we argue, are appropriately situated at the knowledge layer. In that respect, 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.

So, 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 providing services to enhance the Context layer's interfaces.

### 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[9] and Ackerman[1] and in this paper, it needs to support the following set of mixed functionalities.

**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.**



**Data Model** 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 a structured, constrained data model, it should be clear that interaction with the model must be managed exactly as in a modern, multi-user database. In fact, we suggest that implementing a data layer on top of either an object or relational database is the best option for bootstrapping such a system.

**Filesystem View** We have pointed out clearly above that it is essential to support existing applications, work patterns and data formats. As such, it will be necessary 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 cross-walks 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 imply that when connected, a user is interacting with an interactive, shared data model. 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 have observed that granular reuse of data is 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 then we need to provide some sort of universal, granular referenceability *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 synchronization methods.

**Flexible Security and Privacy** Clearly providing a means for users to flexibly manage discovery, use and manipulation of data resources by others 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.

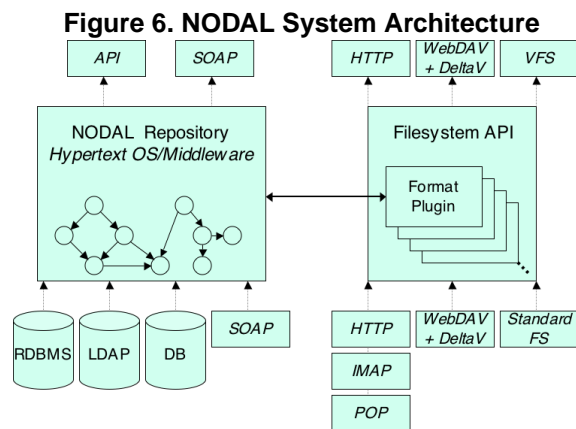
**Search for Content and Structure** For purposes of data mining and information management providing a universal search infrastructure is essential.

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 and as a distributed, synchronizing filesystem. Therefore potential candidates for this aspect of the system would be distributed filesystems such as Coda[15] and Oceanstore[12].

These systems do not, however provide document modeling facilities that would allow them to connect the filesystem view with a database view. There are however, systems that either already implement this kind of facility or are moving in that direction, such as the WinFS filesystem that is part of Microsoft's Longhorn OS initiative[14]. Unfortunately, while certainly integrating the database/filesystem model, both systems use the database elements simply to

replace the metadata aspects of the filesystem model and improve searchability. This is similar to the Presto[8] system developed 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).

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[10], 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.



## 4. Conclusion

Having summarized existing work to identify some of the requirements to create systems to enable effective collaborative work, we identified another endemic techno-

logical 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.

## 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] T. Berners-Lee. The semantic web roadmap. <http://www.w3.org/DesignIssues/Semantic.html>, 1998.
- [3] 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.
- [4] H. H. Clark and C. R. Marshall. *Definite reference and mutual knowledge*. Cambridge University Press, 1981.
- [5] M. Classen. Schema wars: Xml schema vs. relaxing. <http://www.webreference.com/xml/column59/>, 2002.
- [6] C. D. Cramton. The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3):346–371, 2001.
- [7] 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.
- [8] 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.
- [9] J. Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):92–105, 1994.
- [10] L. Iverson. NODAL: Structure and reference for the rest of the web. In *Extreme Markup Languages 2005*, Montreal, Qc., Canada, August 2005.
- [11] 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.
- [12] J. Kubiatoiwicz, 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 AS-PLOS*. ACM, November 2000.
- [13] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages. *Intelligent Systems*, 14(2), 1999.
- [14] J. Rodriguez. WinFS data model. *C# Corner*, Feb. 2004.
- [15] 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.
- [16] The World-Wide Web Consortium. Xml schema: Part 1 (structures). <http://www.w3.org/TR/xmlschema-1/>, 2001.
- [17] S. Whittaker. *Theories and Methods in Mediated Communication*. MIT Press, 2003.