Jeremiah DeWitt Weiner

Prof. Caroline Sdano

Knowledge Management

12 March 2004

The Design and Implementation of a Knowledge Management System for the Field of System and Data Center Administration

INTRODUCTION: DEFINITIONS, EXISTING STATE OF AFFAIRS

This paper aims to cover the design and implementation of a knowledge management system for the field of computer system administration and its encompassing field, data center administration. System administration is the practice of technical setup, configuration, and maintenance of computer and network hardware, operating systems, software, and data, where these things are used in the context of a single operation. A data center is the physical location for a computer network, being physically comprised of the building and its environmental factors and controls (electricity, cooling, physical security, fire-suppression equipment) and more generally comprised of the network as a whole: the computers, networking hardware, external network feeds, and so forth. These disciplines are normally performed by what we can loosely call an operations group, which we can define as a group of technical workers whose primary responsibility is the day-to-day upkeep of the computing infrastructure of the organization. The goal of their activities from an outside perspective is to make technical services available to the rest of the organization; from an inside perspective, it is to do this by performing the operations of selecting, installing, configuring, and maintaining hardware and software, subject to the powers given it and the constraints placed upon it by the organization.

The discipline of the administration of these skills is dependent upon the administrators' knowledge both of what might be called general technical and organizational knowledge, and specific technical and organizational knowledge. For example, general technical knowledge covers such issues as: What is a SunFire E2900? What parts in it are user-serviceable? What is Solaris? How does one create a user account in Solaris? and so forth. General organizational knowledge would include such issues as: What is Unix? What makes it different from other operating systems? In what directions is its development moving? Specific knowledge is that which is specific to a particular host, configuration of an operating system or software, company, or use of a host or software. Specific technical knowledge

might include: What are the hardware specifications of the host "godzilla"? To what port on which router is godzilla's first Ethernet interface attached? Where is the application "eMerchant" installed? Specific organizational knowledge is that which is both peculiar to a specific organization and is not visible in the technical aspects of the system: Who is responsible for the maintenance of godzilla? Who in the organization "owns" the product data stored in the eMerchant application? What is the organization's policy for off-site storage of backups?

All of these types of knowledge are necessary for the smooth and efficient functioning of an operations group in charge of system and data center administration. Typically, operations groups have hired employees with strong general knowledge but no specific knowledge and expected them to learn the specific knowledge by a process of osmosis. There is often an attempt made at documenting and making available the specific knowledge, but it is usually weak, sporadic, and ineffective. (General knowledge is often gained by osmosis as well, via an employee's technical maturation in a group with a strong technical culture and strong general as well as specific knowledge, but this is a cultural issue outside the scope of this paper.) The general failure of operations groups to effectively convey knowledge is surprising, given the strongly knowledge-based nature of most technical work. Eric S. Raymond, in _The Art of Unix Programming_, attributes this to the ephemerality of most computer and programming technologies:

> In most kinds of engineering, the unwritten traditions of the field are parts of a working practitioner's education as important as (and, as experience grows, often more important than) the official handbooks and textbooks. Senior engineers develop huge bodies of implicit knowledge, which they pass to their juniors by (as Zen Buddhists put it) "a special transmission, outside the scriptures."
>
> Software engineering is generally an exception to this rule; technology has changed so rapidly, software environments have come and gone so quickly, that technical cultures have been weak and ephemeral.
>
> (Raymond, 2003)

This ephemerality poses a problem: if information - of any sort - did not exist yesterday, will be less useful tomorrow, and will be only of historical interest the day after, then there is an inherent block to creating a knowledge-storing and knowledge-transmitting culture. And yet the knowledge is no less important at the moment it is needed, for all its ephemerality. Even a worker with strong general knowledge will find it difficult to accomplish much in an environment where he lacks specifics: knowing how to solve a problem will not help him if he lacks permission to solve the problem or does not know how to request permission or of whom to request it, or who will be affected if he does solve it.

THE INTRODUCTION OF A KMS

This author attributes the failure of operations groups to effectively convey knowledge in part to the ephemerality of their data and to the pressures under which they often work, but also to the fact that there has been very little in the way of aids to knowledge transmission. Technical systems are often unfortunately regarded as "self-documenting": "If you want to know something about the system, just look at it - it's all there!" What explicit knowledge storage does exist is often fragmented and incomplete, maintained in an ad-hoc manner. This characterization of the situation is corroborated by other writers on the field, such as Feidhlim O'Neill. Lists of responsibilities, web pages with links to manuals, network diagrams - these things are mechanism, not policy. (A quick definition of these terms is to say that policy is what you do, whereas mechanism is how you do it. Data is that which is dictated by policy and transmitted by mechanism. The fact that a user must authenticate himself to a computer is policy; that he does so by giving a password in a particular way is mechanism; the password itself is data.) In a culture with a historically strong understanding of the separation of policy, mechanism, and data, this is surprising. The situation would appear ripe for the introduction of a knowledge management system.

The environment of an operations group practicing system and data center administration has several features that make it fertile ground for the growth of a knowledge management system. The first and most concrete one is that it is a repository of a great deal of technical skill that can be put to use to build the KMS itself. The underpinnings of a KMS will today almost always be some sort of computer system, and the operations staff will themselves be capable of implementing the technical aspects of the system. This reduces the need for outside assistance and buy-in to the project, and leaves the necessary skills for the administration of the KMS in place in the organization for the time when the KMS will be expanded to the rest of the organization. Another factor encouraging the piloting of a KMS in a

technical group is that at least parts of the sphere of technical workers have strong if often unstated beliefs about the value of knowledge and ownership in knowledge. (Raymond, 1999) This can be exploited to encourage the contributors/users to make their knowledge publicly visible by keeping ownership visible, so that it may feed back into greater prestige. In order for this "prestige mechanism" to work, knowledge contained within the system will need to be associated with its original source, not simply its entrant, who might simply be the "owner" or administrator of the KMS itself.

DESIGN OF THE KMS

Different users will ask for different things from a KMS depending on their skills, their use for the knowledge, and their individual preferences, and different forms of information are best visualized in different ways. A list of users might best be displayed as a simple textual list, a repair history as a timeline, a history of disk space usage as a bar graph, and a network infrastructure as a map. However, some data is searched, indexed, subsetted, collated, or otherwise manipulated in a form other than that in which users would generally like to have the data presented to them. For this reason, it is imperative that data *storage* be separate logically and physically from data *visualization*. This ties in well with the technical culture of operations groups: the culture has a long history of designing extensible protocols and data structures that maintain a strict separation between data, metadata, and transmission and display mechanisms. For example, consider Paul Mockapetris' design for the Domain Name System: a distributed database that is capable of holding arbitrary data in arbitrary data types. The design specifies only the framework, and is mostly silent on the topic of the design is to be implemented (and is necessarily silent on site-specific data). This concept and associated mental discipline is likely to already be familiar to operations staff and management, making it easier to implement a system adhering to this paradigm and to enforce proper adherence in future expansion of the system.

The technical heart of the KMS is the ontology it uses to create knowledge out of the relationships between its component data. An operations department deals with different things: machines, users, software, and so forth. We can begin with a list of data types, the "nouns" in our ontology:

- HOST
- APPLICATION
- PERSON
- PHYSICAL PLANT
- GROUP

- DOCUMENTATION

etc.

The list of nouns should be arbitrarily extensible, so as not to limit the future expansion of the system. Some of them might have subtypes, such as groups of persons, groups of hosts, machine-generated documents, human-generated documents, etc. The other main part of the ontology is what we can call a preposition, (although it does not exactly correspond to the traditional grammarian's view of a preposition), something which expresses the relationship between things:

- owns
- runs-on
- depends-on
- is-member-of
- is-replaceable-by
- documents

etc.

The list of prepositions must be extensible, like the list of nouns. The prepositions connect nouns, but they also carry definitions that determine which nouns they may be used with. For example, what can be owned? Hosts or applications or physical plants, but not people. What can run on things? Applications run on hosts, but nothing else can run on a host, nor can an application run on anything else. So we might express possible relationships in our grammar as follows:

- HOST depends-on PHYSICAL PLANT
- PERSON owns HOST
- GROUP owns HOST
- DOCUMENTATION documents HOST
- PERSON owns APPLICATION
- APPLICATION runs-on APPLICATION
- APPLICATION depends-on APPLICATION

etc.

This structure lets us see and follow relationships: for example,

- PERSON "Richard Roe" owns APPLICATION "eMerchant"
- APPLICATION "eMerchant" runs-on HOST "godzilla"
- HOST "godzilla" depends-on PHYSICAL PLANT "472 West Orchard Rd, Taraville, MD" etc.

This structure lets us see at a glance a chain of dependencies - if Mr. Roe leaves the company, we can see what applications, hosts, and physical plants will be affected. It will not have escaped the notice of the astute reader that the prepositions of this structure imply their inverses:

- owns implies is-owned-by
- runs-on implies runs
- documents implies is-documented-by
- is-member-of implies has-member

etc.

So the relationship between Richard Roe and the "E-commerce Administration Group" might be expressed as:

- PERSON "Richard Roe" is-member-of GROUP "E-commerce Administration"
- GROUP "E-commerce Administration" has-member PERSON "Richard Roe"

Not all objects or categories in the system need be concrete. A document about, "Lessons Learned From the 2003 Upgrade to the Disk Array", could be an object of type "documentation", and could have multiple "about" attributes, so it might be about disk arrays, hence applying to every object of type "disk array", about upgrading hardware, hence in the general category of being about upgrades, and so forth. Financial results, to look slightly further afield, might be their own type, connected to different departments, items on which money was spent, and so forth.

A useful adjunct, albeit one that does not hew strictly to the strict object/attribute ontology, would be a discussion forum, like a blog or Usenet newsgroup. One could be attached to each object or class in the structure (i.e., a forum for host "godzilla", a forum for hosts in general, etc.) although this does raise the question of how knowledge from the forum might be transmuted into a more structured form. This might be a job for the KMS manager, or an automated process could be developed to link forum posts to appropriate objects in the system. This suggests the possibility of a system component which could even learn: it would deduce connections between forum posts and appropriate objects to which to link them, and feedback from the manager or users of the system could strengthen or weaken its "desire" to link similar components in the future.

We can explore the relationships between objects in the KMS by means of their attributes and connections. For example, viewing information about the host "saturn", with the attributes

- depends-on PHYSICAL PLANT "532 W. Orchard Rd"
- runs APPLICATION "MagnaView"
- is-owned-by PERSON "Louisa Sansalvo"

we can choose to view a chain of dependencies, a web-like view (from saturn to all of its connecting objects), or a set of like objects (all things running MagnaView, or all things owned by Louisa Sansalvo). This allows the user to choose a method of viewing which most closely matches his desired aspect, and also permits serendipitous discoveries by exploration: looking at information about host saturn might link to information about host botvinnik, which might link to information about application StatViewer, which might link to information about the Applied Statistical group, and so forth. Keeping the ontology simple and structured will also aid in indexing and searching of the information, and may aid in translation of the information into different languages, just as it would in aiding in "translation" into different graphical views: one can translate the terms on a presentation level without needing to translate anything on a logical level.

It is worth noting that some efforts in the direction of creating a formal grammar for relationships and a means of visualizing them has already been the subject of work both in the computer science field and in certain concrete applications, most notably UML (Unified Modeling Language) and DCML (Data Center Modeling Language), of which DCML is the most immediately applicable.

> "The Unified Modeling Language(TM) (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems."
>
> (IBM)

> "What Is DCML?
>
> DCML provides the first standard model to describe both a recipe and a blueprint of one or more data center environments. Much as a culinary recipe provides both the list of ingredients and the instructions for successfully combining them, DCML

provides both an inventory of data center elements and the desired functional relationship between them."

(DCML Organization)

This author believes these kinds of formal languages are *necessary* for a successful KMS but not *sufficient*; the reason so much time has been spent on the technical underpinnings is that this author believes that it is largely (but not entirely) the lack of such a structure that causes many KMS-like efforts to fail, becoming little more than a folder full of unrelated documents. The primary danger of a system too structured is that it will fail to address the needs of the users and be inflexible and inamenable to change, either when first designed or at some future time as needs expand or change. It is important not to try and answer every possible need, simply because every possible need cannot be anticipated, but rather to make the system flexible enough to encompass new kinds of information so that knowledge can be created from them as well.

(For an interesting take on data visualization, see the "Nosing Wheel" or "Spider Diagram" used to visually display the qualities of Scotch whiskey, an example of a sort of "data synesthesia" not only substituting sight for taste, but visually displaying numeric data in a particular way so as to make relationships within the data clearer.)

HUMAN FACTORS

A system's success or failure is judged by its human users. Regardless of its technical merits or elegance, if it does not answer their needs, only does so at a cost too high to bear, or is more work to use than it is worth, it will not last in use. People can be cajoled, threatened, bribed, or ordered to use a system, depending on the type of organization in question, but enforcement is a tricky matter. Human factors are the most difficult to address and the most common cause of failure of a KMS. "Internal KM experts say that 80 percent of designing KM systems involves adapting to the social dynamics of the workplace. Only 20 percent involves technology as an enabling mechanism." (Hickins, p. 102) "It is our experience that the technical side of information and communications projects is usually handled well. Almost always, the people side of such investments is neglected and causes projects to fail." (Gladstone and Kawalek, p. 244) This section will examine the human elements of the system and how the system can be shaped around their abilities and desires. Humans in the KMS can be contributors, users, and managers, sometimes more than one at a time. Each of these groups (and each person within the groups) will have different needs and wants, and different abilities. Users

would like a system that helps them. Contributors would like a system that rewards them for contributing. Managers would like a system that requires little active management and meets the needs of others so that the system is actually used. Everyone wants a system that is easy to use! Each of these includes a number of subtasks: for the system to help users, the desired knowledge must be in the system, it must be findable, and it must be accessible. These in turn mean that someone or something must have been putting knowledge into the system, indexing the knowledge, providing a facility for searching and viewing the knowledge, and so forth.

For the user, the system should be universally available, easy to use, fast, and applicable. These tie into a great many issues such as networking, security, and user interface design that are beyond the scope of this essay, but it is important to note them all the same. No one will use a system if he does not feel some personal benefit results from it; for this to be true, the knowledge must be applicable. This is a factor for which beginning a KMS with a limited scope such as is described in this paper is a benefit: because the only knowledge in the system is that which pertains to a particular field, it is guaranteed to be at least somewhat applicable. For a future expansion of the system, methods both of interlinking and of restricting information flow may be needed. A KMS which chops up knowledge into unbridgeable domains will not find its full potential, but a user is likely to want to restrict his search for knowledge to a particular domain and not the full knowledge of the company, which might have a great deal of material less relevant to him at the moment of the search. (For a real-life example of this, see Google's "Apple Macintosh", "BSD Unix", "Linux", and other topic-specific searches. (http://www.google.com/advanced_search))

Looking deeper at the contribution side, the reward might be monetary or otherwise concrete, but ultimately it may be more effective and more psychologically effective if the motivation is internal, or can be internalized. "What makes [Xerox Corporation's Eureka] system self-sustaining is the recognition employees receive for participating in the process. ...Holtshouse says, `Recognition seems to [work] the best.'" (Wah, p. 312) This point is particularly germane to technology workers, of whom Eric S. Raymond has said, "In gift cultures, social status is determined not by what you control but by what you give away. [...]abundance [of the "survival necessities"] creates a situation in which the only available measure of competitive success is reputation among one's peers." (Raymond, 1999, p. 99)

The "manager" of the KMS is the person who will be responsible for the day-to-day maintenance of the system and attempts to inculcate a culture of contributing to and using the KMS. In the smaller culture of an operations group (relative to an entire company), it is easier to effect cultural change, since the manager will have closer connections

(culturally, organizationally, and perhaps physically) to the other users of the system. Ideally, the manager will have little to do, but in practice it will probably be necessary for him to add new information, correct existing entries, and so forth.

INDEXING/SEARCH

The object-attribute nature of the KMS described has the potential to make indexing and search much easier, but this is dependent on users entering necessary data properly. Upon the entry of a new object, an automated process could crawl the links between objects, updating the system's index. The object-attribute nature is also a natural fit for the strengths of a relational database, which can provide the underpinnings of a search and indexing mechanism. Searches should be possible both on a full-text basis and on the basis of attributes: about X, owned by Y, reliant on Z, and so forth. The attributes will probably have to be entered manually, but they can at least be added after the original entry of the object.

BUDGET

The underlying design of the system suggests a two- or three-tier architecture: data layer, interpretation layer, and presentation layer. The data layer contains the objects and their attributes, the interpretation layer determines which objects and attributes must be retrieved or submitted and how they should be displayed, and the presentation layer makes the interpreted data available to the end-user. Going the other way, one could say that the user makes requests for a particular view from the presentation layer, the presentation layer passes the machine-readable essence of the request on to the interpretation layer, the interpretation layer determines what must be retrieved from the data layer, and the data layer fulfils the request.

These layers map cleanly onto a model of database, application server, and web server. The exact technologies used for the implementation are not critical, but following existing common practice would make the system more easily maintained by those without specific familiarity with it: it is therefore desirable that the system be composed of commonly available components as far as possible and practical. This would mean a database such as Oracle or MySQL, an application server such as WebLogic or Tomcat, and a webserver such as Apache.

We can make some rough estimates as to the necessary capacity of the system. A medium-sized operations group might have 100 hosts, a staff of 10, and be responsible for perhaps 10 "user-visible" applications and services, plus a number of less publicly visible services upon which the visible ones depend, perhaps 20 more. The hosts of such a group might be located in three data centers, each of which might have a number of services, such as security, environmental controls, electricity, etc.: let us somewhat arbitrarily say there are five such services. On top of these objects, we have a number of meta-objects such as groups; let us say, somewhat optimistically, that there are as many groups as other objects. Let us further assume that each item has five documents regarding it - documentation, manuals, photos, charts, etc. Adding up our objects so far, we have

hosts	100
staff	10
applications	10
app. services	20
locations	3
loc. services	3*5 = 15
subtotal	158
groups	158
subtotal	316
documents	316 * 5 = 1580
total	1896

Now that we have the number of items, we can begin to look at their size. The objects themselves need not be big - most of the data per se will be in the documents regarding them, as the objects are mostly collections of pointers to other things - and much of the knowledge about them can be created on the fly.

For example, webs of interdependencies should be generated at display-time, so that there is no danger of them becoming out of date. If we allocate 1 kilobyte of storage for each object and 10 megabytes (10240 kilobytes) for each document (probably an overestimate, but better to overestimate), we will need

```
objects      1 KB *  316 = 316 KB
documents    10240 KB * 1580 = 16,179,200 KB
-------------------------------------------------
total        16,179,516 KB = 15,800 MB = 15.42 GB
```

This is small by any standard, even if we assume the database requires 100% overhead for each object. Individual hard drives, to say nothing of storage arrays, are already larger than this. To look at data transfer next, let us assume that every user request or submission results in the retrieval of five objects and one document. This gives us a per-request size of

```
objects      5 * 1 KB = 5 KB
documents    1 * 10240 KB = 10240 KB
-------------------------------------------------
total        10245 KB =~ 10 MB
```

If there are 100 requests or submissions per day, this means a transfer of 100 requests/day * 10 MB = 1000 MB/day

Nothing here is even remotely outside the scope of even a very modest server.

For requirements of this magnitude, this author would recommend a modest server well-outfitted with memory and processor capacity, with storage capacity sufficient to hold the entire database, plus room for expansion. Performance will be greatly aided if the entire database, or at least most of it, can be kept in memory at once. It is wise to purchase systems at less than their full capacity so that they may be upgraded incrementally as needed, rather than having to replace the entire system with a more powerful one. A possible server for the demands of this KMS system might be the Sun Microsystems Sun Fire V480 server, outfitted with two UltraSparc III processors, 16 GB of memory, and two 73 GB disk drives. List price for such a system would be somewhere between $20,000 and $30,000. Although the internal disk drives would be sufficient for the starting demands of the database, for performance, flexibility, and future expansion, it would be preferable to put the KMS database and software on an external array, such as the Sun Microsystems StorEdge D2, capable of holding up to 3.92 TB, list price about $5500. Given a margin of error, we can say that the hardware will cost roughly $35,000. A hardware support contract will be required: this is often as much as the cost of the hardware itself, at least to begin, and usually has a recurring component. Software is a different and more difficult matter; the price depends much more heavily on the particular choice of software. There is software available at no charge, if one goes with open-source software, and support costs may be nonexistent if the operations group feels its existing knowledge sufficient to support the software. On the other hand, commercial databases and application servers and the support for them can run easily into the hundreds of thousands of dollars per year. Given that the KMS will not be a huge system, at least to begin with, there is little reason not to go with some of the free

solutions, some of which - such as Apache - are widely acknowledged not only to equal but to beat their proprietary competitors.

EVALUATION OF EFFECTIVENESS

Without an effective metric for measuring success, there is no way in which the investment in a KMS can be evaluated. To come up with a standard, let us return to the business function of the operations group. Briefly, it is to support organization objectives by making technological services available to the organization. It does this by selecting, configuring, and maintaining a technological infrastructure. Therefore, the value of the KMS will be in how it enables the operations workers to do these tasks more quickly, cheaply, and effectively.

A simple metric is how often users refer to or contribute to the KMS, but this is only loosely correlated to how useful they find it. They may be using it or contributing to it out of a sense of duty, or the knowledge that their use of the system is being monitored. An attempt to measure the usefulness of items in a KMS is shown by the number of knowledge bases on the Web which have, usually at the bottom, a short questionnaire, usually saying "How useful did you find this information?" and giving a list of choices. (For example, Sun's online support service, http://sunsolve.sun.com, says on each document, "Please let us know if your SunSolve visit saved you a call to Sun Support!") Something like this can be useful, but if it does not give the user a chance to explain *why* the knowledge was useful or not useful, it is not actionable. This actually touches upon some of the major issues in knowledge management: whether the user is willing to contribute his opinion, whether he really knows why he found something useful or not, whether he can express his views clearly, and so forth.

The effectiveness of the KMS must be judged by its ultimate effect on the work performed by the operations group, and so some of it can be judged by whatever metrics are chosen for measuring the performance of the operations group as a whole: response time, availability, requests serviced, and so forth. The trick is to tell what part of the (hopefully) improvement is due to the KMS. Ultimately all measurements are human measurements: activities must be graded on how well they improve human life. For this reason, evaluation of the effectiveness of a system must finally come down to human factors, as vague and difficult to compare as they can be. More than one user interface expert has recommended watching people as they use your product, to try and see how they actually use it, and what they fail or

succeed at. This might be the single best way to judge the effects of a KMS, because by observation one can see what users are doing as opposed to what they say they are doing, and ask them why they do it.

CONCLUSION

The technical facets of a KMS are the easy ones. As much as designing plans and strategies and weighing the relative merits of different technical solutions seems like "real work", as opposed to the "softer" cultural work of getting knowledge workers to cooperate or figuring out why they refuse, they have answers and they can be completed. The task of inducing cultural change in an organization that has not had a history of knowledge-storage and knowledge-sharing has no single answer and no end. Fortunately for the prospective implementor, the technical culture of the operations team is well-suited for both parts of the introduction of a KMS, and with a successful program in this department, the odds for a successful deployment elsewhere in the organization may be improved.

BIBLIOGRAPHY

Cortada, James W. and John A. Woods, eds. *The Knowledge Management Yearbook 2000-2001.* Boston: Butterworth-Heineman, 2000.

The DCML Organization. *Data Center Markup Language.* 2003. <http://www.dcml.org> 1 March 2004.

Gladstone, Bryan and John Kawalek. "Problems on the Virtual Shop Floor: A Case Study." Cortada and Woods.

Hickins, Michael. "Xerox Shares Its Knowledge." Cortada and Woods.

International Business Machines Corporation. *Unified Modeling Language.* 20 February 2003. <http://www-306.ibm.com/software/rational/uml/> 6 March 2004.

Mockapetris, Paul. *RFC 882 - Domain names: Concepts and facilities.* Internet Engineering Task Force, Network Working Group, 1983

O'Neill, Feidhlim. *Project: Deployment of Knowledge Management system for use in the operations department of "BigInternet Company."* 17 October 2003. <http://www.km-class.org/Fall_2003_Class/casestudies/BIC.htm> 5 March 2004.

Raymond, Eric S. "Homesteading the Noosphere." *The Cathedral & the Bazaar.* Sebastopol, CA: O'Reilly & Associates, 2000.

Raymond, Eric S. *The Art of Unix Programming.* 19 September 2003. <http://www.catb.org/~esr/writings/taoup/> 3 March 2004.

Wah, Louisa. "Behind the Buzz: The Substance of Knowledge Management." Cortada and Woods.