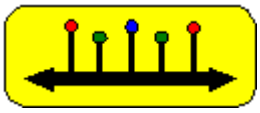

Semantic File Systems

Table of Contents

1.0 Introduction

Motivation for Semantic File Systems

Files are a popular form of data storage as they do not impose any formatting and structuring constraints, and the file/directory abstraction is easy to understand. Conversely, the fact that data is stored in files provides no semantic information about them. Tools that manipulate files carry much of the knowledge about file semantics and form the DBMS for the file formats in a sense. By "file semantics" we mean any higher-level *definitional* (e.g file extension and magic numbers define file type), *associative* (e.g keywords in file that characterize content), *structural* (e.g. physical and logical organization of the data, including intra- and inter-file relationships), *behavioral* (e.g. viewing and modification semantics, change management), *environmental* (e.g creator, revision history) or other information related to the file. Looking at file data through the eyes of tools limits us to the particular "views" supported by the tools. Conversely, tool-independent views of file data will allow a finer-grained, seamless access to file information. It will also facilitate the potential integration of file information with information stored in more structured repositories such as RDBs and OODBs.

This notion of a unified data management framework has received attention in the past few years. The explosive growth of the internet provides compelling reasons for such unification (e.g. more transparent access to all data regardless of source, more efficient querying of file data). The current application view of internet data is fragmented due to the wide variety of protocols and repositories involved. A unified data management framework will allow for seamless, uniform access to this data.

Potential Benefits and Risks

A unified framework for data management facilitates an integration approach to data management in a variety of ways. Some of these are listed below:

- **Interface Integration:** At a trivial level, interface integration refers to a common look-and-feel when manipulating data, regardless of its source. Current internet browsers that support a uniform "folder" view of both local files and URLs provide interface integration at this level. However, there are deeper issues in finding visualization paradigms that would transcend coarse-grained file data and finer-grained DBMS information.
- **Query Integration:** Query integration allows file and database information to be queried as a single integrated information source. As in the case of multi-database federations, issues of data model integration and the definition of a unified query language exist. If anything, the modelling issues are somewhat more severe here than in the multi-database situation, due to the unstructured nature of file data. Some current research focusses on the adaptation of well known query languages such as SQL for querying file information. Clearly, new issues arise not only in query specification but also in the query translation and optimization stages.
- **Reference Integration:** Reference integration allows the modelling, creation and manipulation of relationships in a repository-independent manner. An example of reference integration is mechanisms that allow objects in one repository (e.g a file system) reference objects in the a repository of a different type (e.g. databases).
- **Transaction Integration:** Transaction integration allows ACID properties to be preserved for applications that manipulate both file repositories and traditional database systems.
- **Yet More Integration:** Many other kinds of integration are also possible, such as Namespace Integration, Change Management Integration, Security Integration, Fault Tolerance Integration, etc.

Any attempts at such a repository unification is not without its risks. As with any technology, there is the likelihood that multiple competing approaches to an SFS may crop up, and they will need to interoperate with each other. The task of "unifying unifications" may prove to be even more challenging than the current task at hand. Standards efforts such as OMG and IETF provide one solution to this, in that they form a consensus and limit the number of competing approaches.

Current File Systems

While it's true that current file systems (e.g. Apple File System, Microsoft File System, UNIX File System) do not provide most of the file semantics mentioned above, each does provide at least some limited definitional and perhaps environmental information. The most common file systems today provide definitional information such as file types or magic numbers so that applications can determine if the file content is in a format that they can handle. Most often file types appear as filename endings (e.g. .doc, .rtf, .ps), however there may also be magic numbers embedded within the files that provide a file type encoding. In addition, security-related information can be associated with each file as well as at the directory (or folder) level.

In the Macintosh File System, a file is composed of both a resource fork and a data fork. The data fork contains the file's data as in other file systems. The resource fork, which is an important move in the direction of SFSes, may contain a myriad of meta information about the file, including the file type and file creator, as well as other definitional and environmental information. A file may contain any number of resources, which are typed objects and have a pre-defined structure. Resources could actually contain any kind of information that the application (developers) wished to store there. While it's conceivable that structural as well as behavioral information could be placed in these resources, it is not common. In most cases, data files do not contain a resource fork, but simply a data fork to hold data content (application executables store their code as well as all application meta-data in resources). More importantly, however, the internal definition of each file's resource fork is specific to each application, and it's likely to be proprietary also. Directories (folders) in the Macintosh File System contain the list of files contained within it, the logical location of each file in the folder's window, access privileges, a folder icon (a graphical icon that represents the image of the folder in a window -- each file also can be associated with a specific icon), as well as other information.

Other file systems such as network and distributed file systems (NFS, Transarc's DFS, Sun MicroSystems' WebNFS) also maintain meta-data information in the form of namespace information, security, partitioning, replication, caching information, etc.

While current file systems provide some degree of organized semantic content, they are still far from the extensive, open data model approach that will provide the associative, structural, and behavioral information necessary for successful semantic file systems.

Meta-Data Anyone?

There are also a number of other areas that can benefit from meta-data accessibility. By providing semantic access to this information (perhaps via a semantic file system), other sources will be able to take advantage of its availability (e.g. query, validation, etc.). Some categories of meta-data are listed below.

- **Inter-Data Relationships** - There are a number of systems that provide the ability to define intra- and inter-file relationships, such as hypertext and workflow systems. These systems are responsible for maintaining these associations, which may exist as internal or external links and pointers. These references can be static or dynamic, and uni-directional or bi-directional. As these files move about, change names, etc., these references may need to be updated. In change management systems, when file content is altered, references are followed to provide notification as well as other functions.
- **Runtime Data** - Run-time data is required in a number of areas, such as applications and compilers. Applications require short-term and long-term storage of state information, such as license information, user information, passwords, local license, DNS or SMTP servers, ORB information, etc. (this is equally true of operating systems). Compilers generate information about program data types that is used at compile-time, and perhaps afterward as well (e.g. Lisp).
- **Data Models** - Database management systems, as well as other data repositories such as enterprise registries, the Microsoft Registry, and interface repositories, maintain information about data structures / schemata (e.g. data types, indicies, constraints, relationships, interfaces, etc.).
- **Multimedia** - The aural and visual information which represents multimedia data content, such as images, videos, music, and soundtracks, cannot easily be extracted into forms useful for querying. Therefore, meta-data information is required to provide associative access. In general, this information may be entered by humans who classify the data based upon some set of keywords, or it may be automatically

generated by image processing routines to extract, for example, colors and patterns. This meta-data and it's multimedia content may also be associated with other multimedia data and textual data through relationships, such as provided in CBH.

Clearly the definition of what constitutes meta-data is somewhat open-ended beyond it somehow being "descriptive" of data. However, the widespread use of meta-data to "open up" various forms of data is testimony to its utility.

The rest of this paper deals with one aspect of the unified data management framework. That is the aspect of higher level (object-oriented) models of file information, frequently via the use of meta-data. Such models would reduce the impedence mismatch between file repositories and other more structured repositories, thus facilitating the integration of file stores with other repositories. These approaches are discussed hereafter under the broad umbrella of Semantic File Systems (SFSes). The next section deals with a classification of SFS architectures (the design space), followed by a description of relevant systems.

2.0 Design Space -- Alternative SFS architectures

A design space is a category scheme of design choices that covers a range of variations in approaches. A collection of systems or examples is reviewed and the design space captures the main design dimensions and alternative choices within each dimension. The approach can lead to more general solutions which can emerge from viewing the main design choices. Then, specific systems can evolve toward more general solutions.

Approaches to SFS architectures can be broadly classified into **integrated** and **augmented** approaches. Integrated approaches incorporate extended semantic features directly within the file system. Augmented approaches provide these features via an evolutionary path that augments the traditional file system interface, thus allowing traditional file manipulation interfaces to remain unchanged.

While the basic purpose of an SFS is to provide additional semantics about files, such as file structure information and indexed content, as well as other meta-data, they may also provide more features. These features may include advanced data models (e.g. inter-file relationships), query support, and language support, for example.

Integrated Approach

Integrated file systems present the user with a new and improved file system that is incorporated into existing file systems. These systems provide an integrated data model whereby file data and file metadata are represented within one data model, and the two are implicitly synchronized. These systems (as well as some augmented systems) may provide a type definition language similar to OMG IDL, or some other means to define object structure and perhaps behavior as well. The IDL is used to define the file system's data model so that file content and metadata can be stored together (e.g. file structure, author, revision histories, content indexing, etc.). If structural information migrates from the application to the file system as described above, access routines specific to each file type will need to be defined. These routines (behaviors/methods) must then be made available to applications. While with proper access routines files may be accessed in traditional ways, there will be additional benefits to users and to applications which take advantage of the richer data model. These benefits include:

- Migration of file-related storage to file system - The applications will not need to pack data to store it within a file. The data model can match that of the application's use, and further the data model can optimize it's storage on disk.
- Meta-data availability - Users and applications (such as query engines) will have access to metadata that was otherwise hidden within the data files.

Conversely, integrated file systems may require the user to migrate to a brand new operating system (e.g the x-kernel project from U of A), to a new version of an existing operating system (e.g the layered file system work from UCLA), and to acquire new versions of applications which will access data via the new data model (SHORE, Sequoia2000).

We can classify the level of integration with the current file and operating system as either **loosely-coupled** or **tightly-coupled**. Both types provide a file system with an integrated view of data and meta-data (using the defined data model); however, the loosely-coupled systems sit on top of current file systems (e.g. SHORE) providing a separate file store which is unaccessible via any lower layers of the file system, while tightly-coupled systems are implemented as one or more layers within the file system.

Commercial efforts that fall under this category include Mentor's CAD Framework , which provides enhanced file system services, but requires users of previous versions of the framework to migrate to a new version of their tool suite. Even integrated approaches that maintain traditional interfaces require user data to migrate to the new file management implementation. However, we believe that end-users are conservative with data, especially file data, and view integrated file system schemes as *heavyweight*. Despite these near-term drawbacks, integrated approaches represent the longer-term future of file systems. A particularly interesting approach to building enhanced object-oriented operating systems (which subsume integrated file systems) is an OSA-based operating system. Such an approach is in line with the existing OMG standards effort, and therefore less prone to the risks of completely new approaches. For this reason, OSA-based file system architectures will be emphasized in this section.

Augmented Approach

Description

The **augmented semantic file system** approach provides an evolutionary path to SFS architectures. It leaves the traditional file system interfaces unchanged, while providing a parallel content abstraction view of the file's content. Tools layered on the content abstraction can be more intelligent about querying and manipulating file information. At present, augmented approaches are more prevalent than integrated approaches, as they place fewer demands on the end-user. With this ease of integration comes some drawbacks, which are described in the comparison subsection.

Architecture

Augmented SFSes (ASFS) provide a content abstraction layer on top of traditional file systems to facilitate smarter querying and manipulation of files. Most advanced implementations of augmented SFSes use these content abstractions for either *query shipping* or *index shipping*. *Query shipping* directs a repository independent user query to the appropriate files. *Index shipping* extracts the contents of files and makes them available as meta-data (indices) to a user level query system. The description of architectural components below unifies these two architectures with the caveat that none of these components is mandatory.

Files are abstracted into logical collections, or *domains*. Each domain is managed by a *domain manager*. Domains bear some resemblance to database views in that they are a subset of the semi-structured information space. The word view is avoided, however, as it conjures up notions of conceptual and external schemas which may not be present in a semi-structured world. Domain specifications could be based on traditional OS notions such as directories and regular expressions on file names, or traditional database notions such as view speficication mechanisms. *Domain managers* provide object-oriented views of the file contents in their domains. They also provide the ability to execute queries on their domains. The latter could be viewed as providing query capability on non-indexed attributes of a file. Domain managers can be subdivided into *a content summarization engine*, and a *query engine*. The former is part of the SFS, and the latter is functionality that is part of the QSS(query service) and will be discussed in less detail.

The *content summarization engine* consists of a *file classifier*, a collection of type-specific *file unnesters*, and a collection of type-specific file *content summarizers*. A file classifier infers type information about a file based on implicit (e.g file extensions) and explicit (e.g magic numbers) information. In situations where the file is a composite (e.g tar files), an encoding (e.g compressed files) or both (e.g tar.Z files), the file unnester is invoked to extract the individual files whose content is then individually summarized. The summarization engine executes *content summarizer scripts* on individual files to extract the values of type specific attributes for the particular file type. Examples of such attributes may be header information, author of the file, relationships to information in other files and so on.

ASFSes typically provide a *content exchange language (CEL)* for different elements of the ASFS architecture to exchange information about file contents. Content exchange languages are very loose in their type models to accommodate the weakly typed nature of file repositories.

In an *index-shipping* ASFS architecture, indices and other meta-information is shipped from domain managers to higher levels of knowledge brokers using CEL. This allows the knowledge brokers to directly process application queries. In a query shipping architecture, queries and query results are exchanged between architectural elements using CEL.

Knowledge brokers either direct or indirect knowledge required to process the query. In the latter case, knowledge brokers know other appropriate knowledge brokers and domain managers to talk to. They are also knowledgeable about the required query decomposition, query translation and result collation. Knowledge brokers can be thought of as parts of both the QSS and the ASFS. They perform roles in the latter sense in that they forward (updates to) file content information to other knowledge brokers.

Augmented .vs. Integrated Approaches

Integrated and augmented approaches differ not in their vision of greater access to file content, but in the manner in which they get there. Integrated approaches are the quicker way to get there, if end-users are willing to accept more drastic changes to their operating systems. Augmented approaches accept the OS as is (especially the file system) and provide *less perfect implementations* of the functionality that a user can expect from an integrated SFS, but with almost no negative perturbation to the user's current file access capabilities. At the same time, augmented SFSes can take advantage of progress in the OSes to get progressively closer in elegance and completeness to the capabilities of an integrated SFS.

To illustrate the point about less perfect implementations, augmented file systems may have to work with operating systems that do not notify them of changes to files. They may therefore have to build a custom file polling scheme that is less efficient than file notification. Even after file notification, it would be left to augmented SFSes to detect incremental changes to file content. Here again, an integrated system may be able to modify the file system to support intermediate level content abstractions (e.g. model a file as a collection of text chunks). Another example is providing transactional capabilities in the SFS. An augmented SFS can easily provide transactional capabilities on the extracted meta-data by storing it in a database. It cannot, however, maintain ACID properties across the meta-data and the file data, since it has no control over the ACID properties of file updates. However, if the OS were to support a transactional file system with event notification for file transactions, an augmented SFS could implement such ADIC properties in a manner similar to multi-database transactions.

In summary, while augmented SFSes are tied down by the current capabilities of their OS substrate, they can take advantage of subsequent OS improvements (e.g. less latency in notification of data change, greater access to file content) in much the same way as integrated implementations. As the actual OS gets closer to the idealized SFS OS, an augmented SFS can come closer in functionality and implementation to an integrated one.

3.0 Systems

Augmented Semantic File Systems

Microsoft Cairo

The OLE ***IFilter*** effort is part of the Cairo initiative which provides the text contents of objects to indexing engines. It is analogous to the content summarization engine in the canonical ASFS architecture. IFilter allows for pre-filtering of objects before they are transported across the network. The content abstraction of a file is hardwired to be a sequence of text *chunks.* The file-specific content summarizer in effect decides the chunking policies for the specific file type. One example of a chunking policy might be to use ends of words, sentences and paragraphs to delineate chunks. In the case of embedded documents, it is the content summarizer's responsibility to flatten the document hierarchy into a sequence of chunks by recursively invoking constituent IFilters. An IFilter client is responsible for higher level content abstraction operations such as stemming, full text indexing etc. The interface does not require the object's normal execution context to be invoked. This allows the filtering operation to be relatively lightweight. Currently only Microsoft and standard open data formats are indexed.

State of Implementation: Product

Implementation Platforms: Windows95/NT

References: http://www.microsoft.com/windows/common/contentNTSIAC03.htm

---

Harvest

Harvest in its current distribution is an example of an index shipping ASFS architecture. The domain management function in Harvest is performed by *Gatherer* entities. The domain of a Gatherer is defined in a configuration file using a vocabulary somewhat similar to the UNIX *find* tool, but with coverage of a multiplicity of protocols (HTTP, FTP and so on). An example domain specification could be all nodes accessible recursively from a root node URL. Pruning of hierarchical spaces is achieved by filtering and enumeration mechanisms. The Gatherer includes a type recognizer, file unnesters and type specific content summarizers. All three components are customizable via configuration files.

Harvest Brokers map to the Knowledge Brokers in the ASFS architecture. As Harvest is an index shipping architecture, Brokers (incrementally) pull index information from other Brokers/Gatherers using the brokers *collect* specification.

SOIF (Summary Object Interchange Format) is the CEL for Harvest, and is used to exchange index information between Brokers and Gatherers. SOIF is a fairly simple exchange format that allows weakly typed information to be exchanged as a collection of attribute-value records. A broker's query interface provides access to index information, and may require knowledge about the specific "SOIF schema" (tag names of SOIF attributes) supported by the broker.

Harvest has a large user community, and has significantly influenced products such as Netscapeís Catalog server. In its entirety, Harvest provides most of the components needed for an index shipping approach to layered file systems. Harvest also seems to have given considerable thought to caching and replication issues.

In its current state, Harvest provides very little infrastructure for query shipping. The "collect" interfaces of Harvest brokers are entirely designed to propagate information up from the domains to higher level brokers. Harvest queries are processed using index information stored directly in the broker processing the query. The

absence of a query forwarding architecture makes Harvestís query response only as up to date as the index information in the top most broker. It is conceivable that the Harvest architecture can be modified to deal with this.

Another aspect of Harvest that is adequate for index shipping but not adequate to accommodate other SFS architectures is SOIF, the CEL component of its SFS architecture. SOIF is geared towards the transport of unstructured tuples of (name,value) pairs. Transporting structured information such as composite objects and queries in some intermediate form etc. are beyond the current capabilities of SOIF.

State of Implementation: Source distribution available

Available Platforms: UNIX

Languages: C, Perl

References:

- Hardy, D.R. et al., "Harvest User Manual", U. of Colorado Technical Report, CU-CS-743-94.
- Hardy, D.R. et al., "Essence: A Resource Discovery System Based on Semantic File Indexing", USENIX '93, pp.361-374.
- "Review of the Summary Object Interchange Format (SOIF)"

InfoHarness

InfoHarness falls under the category of SFSíes that uses a OO meta-data repository to abstract(objectify) file, web and some other(ex: WAIS) information. Meta-data abstractions can be generated for both intra-file (e.g: mail within RMAIL file) as well as multi-file data. A couple of simple abstractions (collections and aggregates) can be created on top of extracted meta-data. They did capture some data service aspects in the meta-data, such as the fact that audio files have to be transferred to client machines for handling, while executables are run where on the server. Strange but interesting. Meta-data is extracted on-demand by running InfoHarness scripting language(IRDL) programs.

InfoHarness is closest to Rufus in terms of the SFS functionality provided. Like Rufus, it does not interface to higher level query and resource discovery systems. InfoHarness points out some important attributes of information that the Rufus implementation has not captured. One example is client-server presentation attributes, i.e does the data have to be streamed to the client side or not. These are not, however, limitations in the Rufus framework.

Unlike Rufus which uses an embedded language to encode content summarizers, InfoHarness has chosen to implement an entirely new language(IRDL) for this purpose. The rationale for this decision is not provided, given that it adds substantial complexity to the implementation. Additionally, IRDL lacks the property of interface/implementation separation that Rufus has. This makes IRDL specifications more brittle. Higher-level meta-data abstractions provided by InfoHarness are fairly typical of other SFSes.

State of Implementation: Prototype referenced

Available Platforms: Unknown

References:

- "Putting Legacy Data on the Web: A Repository Definition Language", in WWW3,1995

OLE DB

The objective of OLE DB is to define and implement "an open, extensible collection of interfaces that factor and encapsulate orthogonal, reusable portions of DBMS functionality." These interfaces are provided within the framework of Microsoft's OLE/COM object services architecture also known as ActiveX. From an SFS point of view, it provides tabular access to legacy data stored in file systems. OLE DB can be related to a file content summarizer whereby a *rowset* interface to file content is provided.

Separation of interface and implementation allows OLE DB applications to depend on services without regard to their implementation. An application written using OLE DB to access a spreadsheet would not require modification if the spreadsheet were ported to a relational database. OLE DB is not an open standard; although, Microsoft may open it to a certain degree. OLE DB is implemented on COM rather than OMG's CORBA OSA. The OLE DB interface toolkit consists of interfaces for rowsets, commands, sessions, security, administration, and transactions. The rowset is essentially a tabular cursor or iterator onto a view of the database. Commands are currently limited to strings. An interface to directly import a tree of operations resulting from parsing a string or the output of a query optimizer is described in the literature, but not released yet.

Available Platforms: Windows95/NT

References: http://www.microsoft.com/oledb

Rufus

Rufus is an effort from the IBM Almaden Research Center that provides SFS and query capabilities on file information. While Rufus falls under the category of ASFS, there is little mention in literature of how Rufus can be used to achieved a scalable ASFS architecture. The rest of this survey of Rufus discusses metadata extraction, viewing Rufus as a single domain, single domain manager system.

The client view of file repositories is based on abstract types or *interfaces*. *Interfaces* consist of a collection of method signatures that are *conformed to* by an implementation. Information providers are free to choose implementation objects that represent theire files, as long as they specify the interfaces that these implementations conform to. An implementation can simultaneously conform to any number of interfaces. File instances are mapped to implementation types by the classifier-extractor system in Rufus. The classifier determined the most likely implementation type corresponding to a file instance using pattern matching. Objects conforming to an interface are characterized by *type feature vectors* where features that reveal the type of a file instance may include file extensions, magic numbers, embedded keywords and so on. The *instance feature vector* for a file is extracted from a partially tokenized representation of the file. The interface of a file is determined by finding the interface feature vector that most closely matches the instance feature vector the file instance. Meta-data is extracted from classified files using a class-specific import method.

Rufus can be analyzed in OSA terms (see Role of OSAs). Rufus repositories support a collection of IDL interfaces for different file types. Several Rufus repositories may support a shared set of IDL interfaces via totally different implementation. OSA clients are buffered from repository evolution in that they do not have knowledge of the implementation lattice behind the interfaces. The Rufus *object manager* (one per repository) objectifies the file repository. The object manager could perform other potential functions like mediate between queries to this repository and its objects. A Rufus query could then be shipped over to multiple object managers, that compute query results in ways that are specific to their repository implementation. Rufus has not provided any insight into the process of decomposing and routing queries to multiple Rufus repositories. However, Tsimmis (described above) is the extension of the Rufus substrate to do these sorts of things.

Rufus is unique in its attention to schema evolution issues in a file repository. The urgency to solve the on-line schema evolution problem in file repositories is questionable. In any case, the schema evolution problem does not seem to pose any distinct challenges in the world of files as opposed to object-oriented databases.

State of Implementation: Prototype exists. Source availability unknown. Mention of subsequent use of the prototype in the Tsimmis project.

Available Platforms: UNIX

Languages: C

References: Rufus Show and Tell Page

SFS/Discover (MIT)

SFS implements a file system layer on top of NFS, and supports virtual directories and file content extraction. To maintain meta-data - file content synchronization, all access to files should be through the SFS file system commands (which are NFS compatible). For example, directory navigation commands cause dynamically generated directories to be displayed based upon attributes associated with, or extracted from, the files (file command syntax has been extended to support this). Directory navigation is equivalent to a query on specific file attributes, which in turn, generates an appropriate directory view of the information. In addition, SFS supports transducers, which are file-specific content extractors. The transducers extract file content and associate it with pre-defined keywords. It is these keywords that are used during directory navigation, optionally with supplied values to define a more limited query scope. Transducers support complex files, such as email files, which contain multiple elements. This enables directories to present these elements (sub-file units) as separate files.

State of Implementation: Unknown

Available Platforms: Unknown

References:

- Gifford, et al., "Semantic File Systems," ACM 1991.
- Mark Sheldon et al., "Discover: A Resource Discovery System Based on Content Routing", WWW5 conference

---

Tsimmis:

Tsimmis is a joint project between Stanford and IBM-Almaden Research Center. It aims to provide integrated access to heterogeneous information source (RDBs, Files, OODBs, digital libraries). This review largely focuses on the files aspect of Tsimmis. The primary focus of Tsimmis is a query-shipping ASFS architecture.

Tsimmis does not directly provide any mechanism to specify domains. Some illustrations imply that a Tsimmis repository is a physical rather than a logical repository (e.g. local file system, Sybase, Folio digital library tool). However, there seems to be no such architectural limitation in Tsimmis. Tsimmis uses the Rufus *classifier/extractor* to abstract file content. The classifier/extractor is further described in a separate review. *Translators* in Tsimmis map to domain managers, and *mediators* map to knowledge brokers. Translators are forwarded queries by mediators, and are responsible for optimally executing them. The combination of translators and classifier/extractor maps to the domain manager concept in the canonical ASFSarchitecture.

Tsimmis provides a query language called LOREL (Lightweight Object Repository Language). LOREL is an SQL variant with some added features for querying weakly typed data. Differences between LOREL and SQL include uniform treatment of single and set valued attributes, ability to return heterogeneous sets, wild cards

over attribute variables etc. For queries on non-indexed attributes, translators convert queries into scripts that parse file representations. For example, an SQL "select author from Ö", when performed on a BibTeX file might be converted to a script that examines fields in the file beginning with .AU.Mediators in Tsimmis translate application/user queries and forward to other mediators and translators.

The CEL in Tsimmis is called the Object Exchange Model(OEM). OEM, like SOIF in Harvest allows information to be exchanged between Tsimmis components using a weakly typed (attribute, value) paradigm. OEM is used however, to exchange not only index information but also query results. OEM is therefore more fully featured than SOIF with an ability to capture object identities, and nested objects. The latter is necessary to exchange composite objects between Tsimmis components (e.g between translators to mediators).

Tsimmis mediators have knowledge about other mediators and translators. The kinds of activities mediators perform includes activities such as query translation, query routing, result collation, duplicate result elimination, formatting etc.. Mediators thus coordinate and optimize query execution. Tsimmis provides an environment (partially rule-based) to automatically generate mediators and translators. A constraint management capability that operates on loosely coupled repositories has also been mentioned.

The scope of Tsimmis quite broad, covering repositories from files to databases, and including interactions with "repository agents" such as digital library front-ends. Its direct contribution to the SFS domain is low, as it borrows the classifier/extractor infrastructure from the Rufus project for this purpose. The bulk of Tsimmis'es focus is on the smart routing of queries to appropriate domain. A query translation issue raised by Tsimmis is relevant to SFS'es, namely the translation of user-formulated query languages (in this case an SQL variant) to repository-specific queries (e.g a script that parses a BiBTeX file to extract attributes), is relevant to the domain management layer of SFSes. Tsimmis discusses the architectural need for such query translators and the issue of automated generation of such translators. However, there are no pointers to their implementation. The CEL in Tsimmis (called OEM) provides a solid substrate for information exchange between resource discovery components.

Overall, Tsimmis brings up and discusses a number of interesting issues with regard to SFSes in particular, and resource discovery in general. However, the lack of empirical information on their actual use, makes it difficult to compare these approaches with others.

State of Implementation: Referenced, state of distribution unknown

Implementation Platforms: UNIX, Windows

References:

- Stanford database group publications list

Integrated Semantic File Systems

SHORE

SHORE is a high-performance, scalable, persistent object repository. Since it is an object repository actual data and not simply meta-data are stored within it, as such, the goal of SHORE is to replace standard file systems. SHORE exposes the internal structural organization of files through a language-neutral type system. Object types (attributes, relationships and behavior) are defined using SDL which is an extension of OMG's IDL (adding features such as sets, lists, and persistence). The type of each object is specified by including a pointer to its type object (e.g. akin to a roles approach).

SHORE provides internal structures to represent "UNIX-like" hierarchical name spaces providing the

appearance of a standard file system (including directories and symbolic links, as well as "pools" and cross-references). In addition, a SHORE repository can be NFS-mounted and transparently accessed by any legacy applications. When an object is accessed by one of these SHORE-unaware applications, the repository returns the content of a single text attribute (a variable length byte or character string) representing the "Unix data" component of the object -- as long as the defined object model has included such an attribute. Ideally, appplications will be designed to take advantage of the type-specific file information defined in SDL to access the data in an enhanced manner. For scalability SHORE uses a symmetric, peer-to-peer distributed architecture, so each server must run a SHORE server process. SHORE supports locking, distributed transactions, logging and recovery. They are currently exploring query mechanism support.

Due to the structural integration of data and meta-data, partial file locking based upon structure (fine-grain file referencing) is possible. In addition to defining file/object structure, SDL can also be used to define inter-file relationships. And, while it provides both DBMS as well as file system features, enabling access via legacy systems, this access simply provides a migration path for current applications. The full benefits of the system can only be realized as applications are modified to access file data via the type system and operations.

State of Implementation: source code is available. Latest version 1.0 from 8/95

Implementation Platforms: Solaris Pentium/Sparc

References:

- SHORE home page
- Shoring Up Persistent Applications(.ps.Z), ACM SIGMOD 1994.

Synopsis File System (SynFS) -- Transarc Research Group

SynFS is offered as a proposed layer on top of Transarc's distributed file system products. While described as an experimental wide-area file system because it sits on top of a distributed file system, it is better called a semantic file system as it doesn't add any specific features to augment the underlying file systems to support distribution. SynFS defines file metadata information as synopses which are organized into digests (e.g. views). Synopses are typed file objects within SynFS and they are defined using TDL (Type Definition Language). TDL supports the specification of attributes, indexes and operations, while operation code is defined using TEL (Type Extension Language), a scripting language based upon Tcl. Digests can be defined using query predicates based upon type information.

Using a layered approach, where SynFS maintains file surrogates (synopses), requires that the synopses be updated in conjunction with file updates. SynFS supports this "auto-indexing" feature. Inexes can be kept synchronous with file updates if callbacks are supported, but it is expensive to monitor all file activity. They are devising an approach to support payments for "services rendered" with respect to consistency maintenance.

Digests represent user-defined views of the system allowing different semantic organizations of the information. And, unlike most other augmented approaches, SynFS supports auto-indexing of meta-data synchronous with file updates (partially-sychronized). However, the maintenance of this synchronization can be expensive. Locking level is the entire file.

State of Implementation: 1.0Beta (.tar.gz) -- January 1996

Implementation Platforms: Solaris 2.4 (DCE 1.1 required)

References:

- **Transarc Research Group's** home page
- Publications

Zoomit

http://www.zoomit.com/white.htm

Zoomit is currently working on a meta-directory service to sit on top of current file systems. The primary feature of their (future) product is the concept of a "join", like a relational join. The join contains objects which represent a cross-referencing (or joining of all related files, messaging/groupware applications and databases in the system). Changes to the meta-directory, or to the referenced information are propogated in both directions to synchronize both. To accomodate this, relationships are defined between objects across name spaces. A graphical modeling tool is used to express and alter these relationships.

This work is interesting in part because it seeks to merge data from multiple types of data sources (a federated systems approach), and as it provides bi-directional update propogation (unfortunately details of this mechanism have not been described). They propose the use of a high-level graphical tool to define relationships between data sources.

State of Implementation: Not Available

Implementation Platforms: Unknown

References: http://www.zoomit.com/

Semantic File System Components

O2Yacc

O2Yacc is a prototype implementation of a file data structurer built on top of the O2 OODB product. While it does not have all the components of an SFS, it deals with the critical question of modelling and querying the contents of a file. O2Yacc aims to express the structure of file types as grammars or ***structured schemas***. O2Yacc then parses the file using the structured schema and "loads" it into a database structure. While traditional context-free grammars are clearly adequate to parse files, they do not provide the context of how to unambiguously translate file contents into a database schema. Such ambiguities are removed by means of an annotated grammar. Letís say that a file containing a list of names can be expressed as a simple Yacc list rule. There is ambiguity on whether this list manifests itself in the database as a set or some other data structure. Rule annotations make this translation unambiguous. The "loaded" database can now be used to query file contents more intelligently using traditional OO query languages.

Clearly, exhaustively "loading" every file in the information space is expensive and quite wasteful. O2Yacc discusses an on-demand parsing scheme where queries to an (unparsed) file are "pushed" into the grammar specification, thus parsing the entire file, but loading only to the extent required to process the query.

There is a potential synergies between resource discovery systems like Harvet and Tsimmis, and O2Yacc. O2Yacc may be able to hook into Harvest/Tsimmis for files whose content needs to be abstracted in rich ways.

State of Implementation: Prototype referenced. State of implementation and code availability unknown.

Available Platforms: UNIX

Languages: C

References

- Abiteboul, S. et al., "Querying and Updating the File", in Proc. Of the 19th VLDB Conference, 1993, pp. 73-84

Semantic File System Infrastructures

Spring

The goals of the *Spring* project were to design a modular, extensible Object-Oriented operating system. Two major design decisions of *Spring* were a micro-kernel approach, and the use of ORB-like mechanisms to model higher-level OS components and their interfaces. This description concentrates on the latter aspect, which makes *Spring* an interesting model for an SFS.

OS components that are traditionally hardwired such as file systems, cache manager, and virtual memory subsystems are extensible, replaceable and layered in *Spring*. A *Spring* instantiation can therefore support both multiple file systems as well as interpose a new file system that uses (and build upon) other existing file systems. File systems are therefore *file object managers* file management operations, *and file objects*. All this is achieved by adopting exactly the same philosophy as CORBA, namely - *language-independent interface definition, language specific implementations*, and the *subcontract* approach to client-server agreement which specifies further run-time aspects of the binding.

While Spring does not in itself concentrate on content abstraction of file information, the architecture is entirely reusable for doing so. Further, it provides a path for marrying the OSA approach to operating system design with all the implied benefits. Attributes of file objects could be derived from content rather than from typical *inode* information. The file object manager could potentially turn into a COSS (Common Object System Service).

State of Implementation: near-production quality

Available Platforms: SparcX (Sparc2 and Sparc10 referenced)

References:

- Spring research group publications

UCLA File System

The file system approach described (likely related to the Ficus project) outlines the design of a layered file system. A number of requirements are identified that they claim are necessary of any layered approach:

- **Extensible** - the system must be extensible enabling new operations to be added.
- **Stackability** - Linear and non-linear stacking should be supported. Layers provide *pass-thru* extensible interfaces -- if an upper layer doesn't understand a lower layer's commands it simply passes it through using by-pass routines. Finally, each layer is assigned a name in the file system allowing potential user access (a complete file system is created by successively mounting individual layers). Stacking also encourages reuse of individual layers.
- **Well-Defined Interfaces** - Layers require well-defined interfaces, including meta-data about operations.

- **Rich Flow of Control** - calls should be supported in both directions through the layers (e.g. for cache consistency in distributed environments).
- **Opaqueness** - The operating system must limit itself to accessing the files and structures through the interfaces provided.

The layered approach to file system design is conducive to the design of semantic file systems. Different layers of the file system could provide different kinds of semantic file information. A layered approach to file system design could even help facilitate the implementation of a global object unified data management framework by providing layers which could be used to access/query other data repositories.

References:

- Hiedemann, Popek, "A Layered Approach to File System Development," UCLA TR-CSD-910007
- Related Paper: Heidemann, "Stackable layers: an architecture for file system development", UCLA TR-CSD-910056

Other Meta-data Systems

Content-based Hypermedia (CBH)

CBH is a database system that stores meta-information about multimedia data such as images. As multimedia data is entered into the system, the user provides additional meta-information, including the establishment of new relationships with other meta-information and other multimedia data. This is the means by which semantic information is generated within the system. For example, when an image is entered the user can identify specific regions of the image as "semcons" (iconic data with semantics), which then become objects themselves, and then attach specific descriptors such as shapes, textures, color, or other content descriptions (e.g. for a semcon containing an image of the nose of Bill, it might be called "Bill's nose"). The user can then associated the semcon with other semcons and other multimedia data. These associations provide the ability to browse the database based upon semantic content. In addition, they investigate support for navigating using shaped-based similarities and spatial relationships. Finally, the system creates "clusters" which represent individual access patterns over the metadata allowing the user to define views without affecting the underlying data.

This system provides high-level browsing of multimedia information and user-definable access paths for browsing optimization. At this point, however, there is no support for maintaining metadata and data synchronization (change management). Data entry also requires significant time investment, and is error-prone as it relies either upon limited categorization schemes or subjective categorization. Finally, the metadata features list cannot be infinite in length, and therefore, some needed categorization of the data will not be available (data can only be retrieved using the described categories). However, these are concerns of most similar systems.

References:

- William I. Grosky, "Using Metadata for the Intelligent Browsing of Structured Media Objects," SIGMOD Conference, 1994.
- Multimedia Lab at Wayne State University (headed by Grosky) -- CBH information may appear here in the future

---

4.0 The Role of OSA in file systems architectures

There are at least two possible scenarios for the interaction between OSAs and FSes:

- OSA as an implementation substrate for FS

- FS as a service(COSS) within the OSA architecture.

OSA as an implementation substrate

A FS is a complex multi-process, distributed application that is expensive to build and extend if constructed using traditional programming techniques. An OSA could be viewed as providing an object-oriented operating system kernel on top of which higher level applications reside as separate objects or object managers. The well known benefits of OSA architectures, namely strongly typed interfaces, language-independence of interfaces, extensibility based on interface/implementation separation etc. enable the construction of extensible and maintainable FSes. The layered file system above, for example, could be viewed as a collection of interacting typed object managers for functions at all levels. *Untyped Files* could be thought of as objects satisfying the *untyped file* IDL interface for general purpose file operations. A *file classifier* object manager is passed *untyped file* objects by the *content summarizer* object manager, and returns *typed file* objects, where the types of the file are determined using some classification heuristic (file extension, magic numbers, identifying keywords within the file etc.).

A number of the projects surveyed below can be extrapolated in a fairly straightforward manner to OMG vocabulary. Spring's adoption of an IDL, the notion of object managers and the notion of services provides significant parallels. Others like Rufus that use the notion of file objects and notions of separating interface and implementation also provide parallels.

Encapsulating File Systems as an Object Systems Service

Regardless of how the FS is implemented, there is a motivation for a COSS that allows OSA applications to view FSes as an objectified repository whose objects collectively and individually support strongly typed, semantically rich interfaces. Such a semantic file repository service (SFRS) has significant ties to a number of existing services such as the persistence service and the externalization service.

The SFRS has much in common with OMG efforts (such as the OMG Persistence Service and the OMG object externalization service) in that it attempts to impose a stronger, richer type system on file repositories than is currently imposed by OSes. The OMG COSSes in their current incarnations model file structure rather than content. Given the shared architectural framwork, however, there is significant motivation to incorporate SFRS needs in a subsequent version of an existing COSS, or an entirely new one.

Another potential marriage of OMG ideas and SFS ideas is in the notion of a CEL(Content Exchange Language) that was discussed in the layered file system section above. The persistence service has a need to transport information in a repository neutral manner. In fact, the Dynamic Data Object Protocol (DDO) proposed in the Persistence Service (pg. 143) bears significant resemblance to the CEL in one of the projects reviewed below (namely the SOIF component of Harvest). It is conceivable that the notion of a repository-neutral information exchange language has to be elevated beyond the OMG persistence service into a facility of its own. An enriched version of such a language can serve the purposes of both the persistence service and the SFRS.

References:

CORBA Services Specification Page

This research is sponsored by the Defense Advanced Research Projects Agency and managed by the U.S. Army Research Laboratory under contract DAAL01-95-C-0112. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency, U.S. Army Research Laboratory, or the United States Government.

This page was written by Venu Vasudevan and Paul Pazandak. *Send questions and comments about this page to venu@objs.com or pazandak@objs.com.*

Last updated: 1/3/97 sjf

Back to Internet Tool Survey -- Back to OBJS