An Essence Store for Video Post Production

David Stephenson, Russell Perry

HP Laboratories
HPL-2008-181

Keyword(s):
storage service, essence media

Abstract:
This paper describes a new reliable storage service called the Essence Store, which was developed as part of the HP Digital Media Platform. The service manages large file copies in a reliable and scalable manner and supports integration with a range of different storage systems including hierarchical storage management systems and off site storage locations. Central to the service architecture is a job abstraction for file copy operations and a tiered model of storage. The Essence Store is a storage service tailored to the needs of the post production studio environment, but could be deployed in another application area where large file management represents a significant challenge.

External Posting Date: October 21, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: October 21, 2008 [Fulltext]

An Essence Store for Video Post Production

David Stephenson

Russell Perry

Hewlett-Packard Laboratories

Bristol, UK

1 Introduction

This report describes a storage service, which was developed as part of the HP Digital Media Platform. The intent of the service is to support developers building post production solutions such as our partner Ascent Media Group [1]. The storage service is called *Essence Store*, so named because "essence" is a commonly used term in the media industry to refer to the bits behind a media artefact such as a video, audio or image.

Post production organisations provide video content owners with a wide range of services. Services of particular importance to the development of the Essence Store are media asset storage management and creation of variants of owners' media assets. These variants include foreign language dubbings, edited versions for aeroplanes or specific encodings for digital delivery or sale. The master and derivative media asset are held on a digital storage device and manipulated in the digital domain on industry standard servers and storage devices.

Essence files and media assets in the post production environment are typically very large and post production will often involved working on sets of assets belonging to a single owner as a batch order. This presents many challenges for standard IT storage infrastructure and software which has different design centres such as archival of email and office documents. The storage service is required to support petabyte storage consisting of many large media files, concurrent ingestion of multiple large files at high aggregate throughput, controlled placement of files on storage media to support low latency access and integration with workflows. Additionally the service needs to be easily scalable and fault tolerant since it underpins the key post production business processes.

This report makes numerous references to "StorNext" which is a Hierarchical Storage Management (HSM) product. StorNext is a trademark owned by Quantum Co.

1.1 Content Workflows

Post Production Houses provide many services to the film and TV industry; several of these services are high-touch and craft based, while others are more amenable to software automation. Post production houses now provide digital workflows services to their customers; these services include storing the digital media assets in very large tape storage facilities and producing variants of these assets on demand. There is now a growing desire amongst the customers of post productions houses, for example movie studios, to be able to reduce the post production costs by leveraging standard IT tools and equipment. While film and TV have actually been captured and stored digitally for some time, the tools used to manipulate this digital media have until recently been hardware based. Now that increasing numbers of conforming and transcoding tools are compatible with standard IT server environments, there is now the ability to both store and process media within an IT data centre environment.

Post production workflows come in four broad classes, which in reference to an asset relate to the stages of the assets lifecycle. These are *ingest*, *manage*, *process* and *export* and, are illustrated in Figure 1,.

Figure 1 : The four broad classes of workflow

In the sections below the four classes of workflow are each elaborated on with particular reference to how the Essence Store is used. The workflows described have been greatly simplified for brevity.

1.1.1 Ingesting a new asset

In post production a media asset denotes a collection of digital elements that collectively makeup a single customer item. For example, a movie asset would include the master, trailers, marketing material, audio elements, and could include foreign language versions etc. Media assets need to be first *ingested* into the post production solution and stored before they can be used in further workflows. Assets may be made up of several “essence” parts and metadata information. A range of mechanisms need to be supported for ingesting essences into the solution. This includes reading files from local or remote ftp staging servers, reading from portable disks, or from the local disk of an operator’s client machine. Metadata records may be manually entered, or automatically extracted from the essence files, after ingesting the essence files. Multiple tape copies will usually be made of the high value essence files.

1.1.2 Processing an Asset

Processing workflow stages cover a wide range of activities. For example, generating a new rendition of an asset such as an encoding of a movie trailer compliant with Apples format specification for video. This process may be automatic, as would probably be the case with the trailer for iTunes, but may include manual steps for example when creating a rendition for an airline where editing is required to remove some scenes. Irrespective of whether the process is manual, or automatic, the master essence will almost certainly need to be moved from tape storage to hard disk for processing.

An essence may well be stored on one or more tapes that may have been removed from the robotic library cabinet (see section below on HSM systems for more details on tape movements). To access the files containing the essence, the tapes will need to be moved from their current locations back into the robotic tape library and from there have the essence bytes restored back onto hard disk. Movement of essence files, therefore, needs to controllable by the application which has prior knowledge of future work orders. This is supported by a leasing model, which allows storage space to be reserved to hold essences on a given storage tier for a specified amount period of time.

Once copies of the essence files are available on disk storage, the application can then initiate the processing steps that are required. The results of processing are typically new essence files that will need to be imported into the Essence Store. After all the processing steps have completed, which may take only a few hours or could take days, the application releases any associated leases on the storage that were being used to hold the master essences. Internally to the Essence Store, the new essences will be copied to tape, if configured by the storage policy, before they are truncated (see section 1.2) from the online disk storage.

1.1.3 Exporting an Asset

Once new variants of an essence have been created and stored they will need to be exported to a storage location for packaging (e.g. combining metadata and essence files into an archive file) and final delivery to the asset owner. This might be the logical conclusion of a work order to create a new rendition or the asset owner may require the essence to perform their own processing.

The application will arrange for the required essence files to be moved onto a disk provided as a file drop location. Typically, at request time, the essence files will be held on tapes stored outside a robotic library. Therefore, the tapes will first need to be moved from their current locations back into the robotic tape library and from there the essence files can be restored back onto disk. The application then arranges for the essence files to be copied to the external location and when completed successfully, the disk storage used to store the essences can be reclaimed.

1.2 Storage - Hierarchical Storage Management (HSM)

Hierarchical storage management usually refers to systems that trade access latency for cheaper storage costs. Two or more levels or storage (or *tiers*) form a hierarchy where the frequently and recently accessed data is stored on faster (but more expensive) hardware and rarely accessed data is kept on slower but cheaper storage hardware. The access latency for a file may well not actually be

any slower if the file is already cached on the low latency tiers; if file access patterns follow normal IT usage then most accesses will experience normal latencies. There are several possible storage hardware options including: solid state disk (SSD), hard disk drives (HDD), optical disks (e.g. DVD-ROM, laser disk) and magnetic tape. Generally HSM systems include software that support standard POSIX file system semantics (open, read, write, seek) over all the files in the system, the faster storage layers (e.g. disk) are used as a cache over the slower storage layers (e.g. tape). If a data file is opened for reading and that file is not currently on the disk layer then the file is copied from tape onto the disk where the file can be directly accessed. The HSM system monitors the way data is accessed and attempts to optimise which files can safely be moved to high latency tiers and which files should stay on the low latency tiers. When the disk space exceeds a specified watermark, the HSM system will start to free space on the low latency storage by *truncating* some files from the disk. Truncation removes the file bytes from the disk, but unlike deletion the file still exists on the file system. Truncation can only be applied to a file once the required numbers of tape copies have been made of that file and sufficient time has elapsed since the file was written.

HSM software typically allows configurable policies to control how files are managed. Policies may cover

- How many file copies are kept (by name, size, type)
- Which storage tiers files are stored on
- How long files are kept in low latency layers before being eligible for truncation.

Different policies can be setup for different parts of the file system thus enabling tailored storage characteristics appropriate to different applications and user storage. Although HSM systems maintain an appearance of a large random access file system there are practical limits to this façade. Critically, access to a file held on high latency storage tier may well take several minutes to be copied to a low latency storage tier that allows random access. Although operating systems give no guarantees about how long file read operations take, many applications and services often expect sub second responses and may well fail when presented with several minutes of delay. HSM systems may be configured to automatically create multiple copies every file.

To achieve cost savings, HSM systems are configured to store large percentages of data on higher latency but cheaper tiers, typically made up of large numbers of optical disk or tapes which are held inside a robotic library. The library can contain several drives to allow multiple concurrent read and writes. The library also contains a mail slot for inserting or removing media from the library to store on shelves. Each individual optical disk or tape has large, but limited storage. A very large essence file can in fact span multiple tapes, although usually 2-3 large essences will fit on to a single tape. Robotic tape libraries typically hold hundreds or thousands of Linear Tape-Open (LTO) tapes, giving aggregate capacities of a few petabytes.

While a HSM system can have an arbitrary number of tiers or hierarchical storage, there are three important classes of tier that have specific characteristics:

- **Online**: the online tier provides immediate random access to the file stored in that tier, can be implemented as hard disk or RAM.

- **Nearline**: the nearline tier provides automatic access (i.e. no manual intervention) so that a file can be moved to an online tier or accessed directly after some delay to move the media into a tape drive for reading. This tier can be implemented as a robotic tape library or as a robotic optical disk library.

- **Offline**: the offline tier provides manual access to the files stored on the media held on this tier. The user knows to either move the media to a nearline location, or the HSM software requests such a movement when the file access if first attempted. Media removed to a remote location in order to enable disaster recovery would be an example of offline storage.

1.3 Customer Need / Impedance mismatch

As the digital enabled post production business has grown, the number of movie and TV titles that are stored and the number of new titles that need to be ingested per month has increased. This increase puts strain on the standard IT infrastructure. By way of an example, each movie title is stored and processed as a compressed MPEG-2 file which is typically very large, on the order of 80-180 GBs. The master file must be pulled from a staging storage area and "ingested" into the service. The average customer can have hundreds of titles, so the ingest rate on the service can exceed 15 TBs *per day*. To handle this capacity, HP StorageWorks EVA SAN storage arrays coupled with HP StorageWorks ESL E-Series Tape Libraries have been used to provide a cost effective hardware base for solutions. The StorNext Hierarchical Storage Management (HSM) file system from Quantum (previously ADIC) is used to control the EVA and robotic tape libraries. Generally each asset (title) is copied onto three tapes to ensure redundancy; one tape is moved offsite and the other two are generally kept outside the library. Only when the essence is required in a workflow will the tape that contains it be put into the cabinet.

HP worked with partners and customers to determine the current limits of existing hardware and software solutions in the post production domain. The following pain points are illustrative of the issues raised.

- The aggregate ingest rate could, if unconstrained, exceed the HSM's ability to write copies of the new files to tape.

- The operational processes for moving tapes out of the library and onto shelves (and the reverse) were very operator-intensive and did not scale.

- There was no view of the physical location of an file in the storage system.

- Latency when reading files can be in the order of hours causing workflow scheduling inefficiencies.

- There is no means to control the load on the storage system and specifically to limit it when there is a peak in activity. No queuing or back-off behaviour in face of large number of requests.

- Lack of fine grained control over how files are truncated by the system.

- A single HSM policy configuration did not work for all media types and sizes.

- No programmatic interface to control the HSM system was available, prohibiting any solution “smarts” to address the above problems.
- When all tapes containing copies of a particular file were removed from the library (due to manual errors) then access to the file fail or never complete.

The issues listed above arose primarily because the data access patterns exhibited by the solution were very different from the normal file system access patterns seen in other deployments of HSM systems. Also the disk tier was heavily utilised by ingestion processes. These often lead to congestion and undesired caching behaviours, which in turn lead to application errors, unexpectedly long job completion times and low total throughput. This situation was compounded by the fact that the robotic library behaved analogously to a second level cache, when the robotic library become full, tapes are removed to create space, and placed on a shelf. It was often the case that all the copies of a file were on tapes outside the tape library, this then caused any file access to fail if the manual tape insertion to the robotic tape library did not occur very promptly. Manual processes were required so that the required tapes were moved from offline locations back into the robotic tape library before initiating an application request for an essence. This manual pre-fetch step was complex. Determining which set of tapes held the essence, finding the tapes, ejecting tapes to make space, and finally inserting the required tapes into the correct tape library and all this was done outside the scope of the solution.

1.4 HP Solution

HP, working with partners, agreed an interface and model for a new service that would meet the application challenges described above. The service model defines the key concepts to abstract various storage systems and operations and is described in section 2.

Since file sizes handled by the Essence Store are very large, file movements can take a long time to complete. For this reason, a *Job* abstraction was introduced to capture the status and progress of long duration activities. In fact, the operations supported by the service interface can be divided into those that are of short duration and those that are long duration which will always be managed as Jobs. Both types of operation, respond synchronously, but in the case of long duration operations, the activity associated with the operation is not complete until the job completes. As is typical in other systems, there is an event model associated with the job to allow tracking of job progress without requiring polling.

A web service interface for the new storage service was defined. The interface was described using WSDL with a SOAP binding as the applications using the new storage service used SOAP as the common means of interaction. SOAP and WSDL are also broadly supported by many platform providers and allowed the architecture to benefit from other developments in the WS stack such as WS-Security [3].

A critical part of this work involved working with ADIC (later acquired by Quantum) to refine an HSM API that would allow the Essence Store software to directly control the HSM layer and query it for the location of tapes, control tape movement, etc. Previously, the only way to interact with the StorNext HSM file system metadata controllers was through a browser interface or Unix style

command line tools. The Essence Store uses the HSM API to control the operation of the StorNext software. This API can only be invoked on the metadata controller in the StorNext system, and not from just any client that had the file system mounted.

1.4.1 Essence Store Features

The Essence Store supports the following high level features:

Segmentation of Assets: Different types of Essence files are placed in different Essence Store "Silos", or logical file stores. Each Silo can have a different storage policy associated with it customised to its use.

Support for different Storage implementations: The Essence Store has been designed to support different underlying storage systems including HSM systems and hard disk only storage.

Metadata aware: The Essence Store is not intended to be a metadata store, but allows name value properties to be associated with an essence. Much of the metadata related to an essence is provided by the underlying storage system. This is exposed in an implementation neutral format using reserved property names. In addition, the metadata model defines certain other properties describing silos, locations, media and tiers.

URL Access: The Essence Store allows essences to be read and written using URLs. A resolvable URL for reading or writing is referred to as an *accessor*. Currently HTTP and FTP accessors can be obtained for essences. This is a convenient access mechanism for smaller essences. For large essences, it is recommended to use the import and export functionality (section 8.2.2). Accessors can be hosted by separate HTTP and FTP Server heads.

Job support: All operations that invoke long duration activities result in the creation of Essence Store jobs. The Jobs provide a means to manage and track the progress of a long running activity. The operations that result in the creation of jobs are detailed in section 2.1.2. Job completion is notified asynchronously through events if a callback address is provided when an operation is invoked.

Event support: Some events are naturally related to the progress of Jobs. Other events provide notification of lifecycle events associated with essences. The set of events supported by the Essence Store are described in section 8.4. The events can only be sent to an HTTP URL callback address. The event model is simple, and is broadly similar to WS-Eventing and could be readily replaced as and when mature implementations of WS-Eventing become available.

Tier Management: The Essence Store exposes which storage tier, online, nearline, or offline a particular essence is being stored at that moment. It also provides control over the tiered placement of essence is the silo containing the essence has multiple tiers.

Essence Leasing: Leasing is closely related to tier management. The Essence Store defines a leasing mechanism such that essence files will be guaranteed to be available from a given tier over some period of time. This also allows for better management of essences to minimise access delays during operations (e.g. pre-emptive placement of an essence onto disk just in time for processing by an editor).

Fault tolerance: the Essence Store was designed so that hardware and software failures would not lead to loss of state, or failed jobs, due to single hardware or software failures. For large files, TCP connections it is not uncommon for the connection to fail. Given the time involved in writing files it is vital that the Essence Store automatically deals with such communications failures as efficiently as possible.

Scalability: The Essence Store implementation should be deployable on a cluster of servers to provide scalability primarily for higher input and output bandwidths. The Essence Store also supports connections to multiple tape libraries so that nearline tape storage can be scaled as required.

Intelligent throttling: The Essence Store will limit the number of concurrent jobs executing so as to avoid failures due to overloading of tape decks and buffers.

2 Storage Service Description

This section describes the design and implementation details of the Essence Store.

The Essence Store provides a generic storage management model and interface over one or more storage management products or systems. The Essence Store's primary function is to support the import and export of essences, or content, into specified storage silos and to control the placement of essence files on a storage tier appropriate to the needs of the application.

The Essence Store exposes a Web Service interface to manage importing and exporting of essences. However, an essence is not directly imported /exported through the web service, the import operation is used by an application to specify a source URL location from where the essence can be imported; the Essence Store will resolve the import URL and upload the essence. Conversely, for an export operation, a destination URL is specified.

Guaranteeing that an essence is available for immediate access by a solution for some amount of time can be very helpful to minimise the time that a user spends waiting for an essence to be accessible. The Essence Store includes a content leasing mechanism that solutions can use to control the availability of an essence on a particular tier.

2.1 Service Interface

This section describes the operations provided by Essence Store interface.

The currently supported operations are classified as being of either short or long duration. Short duration operations are effectively completed as soon as the synchronous response has been received by the caller. Long duration operations will respond synchronously to indicate that the operation has either started or failed to start. However, the completion of the operation will take a period of time to complete which is dependent on the type of operation being performed and other issues such as the size of the essence (content) and/or the load on the system.

When long duration operations are invoked, they are initially checked for correctness and if there are no outstanding actions on the related essence then they are stored in the database as "*jobs*".

Note that the response from the initial import/export invocation only indicates success or failure of job creation and not completion. The import/export operations do not necessarily begin immediately. As and when resources are available, the import/export jobs will begin. The number of imports/exports that can be run in parallel from a single instance of an Essence Store can be configured by the solution administrator. Progress of an on-going long duration operation can be tracked by the application that initiated the job by subscribing to the series of events generated by the Essence Store.

Each of the operations returns faults in the event of an error executing the request. There are several common fault types that can be returned such as (generic) service fault, authorisation fault and message parsing faults. There are also more specific faults such as invalid parameter Fault and unknown identifier faults for all the identifiable entities in the Essence Store model.

2.1.1 Short Duration Operations

CreateEssence

An essence is created by calling the CreateEssence operation. This will require an essence identifier as well as the silo in which the essence will be stored in. Once created, it is possible to import content or obtain an accessor to write content into an essence.

DeleteEssence

When essences are no longer required they can be deleted using the DeleteEssence operation.

SetEssenceMetadata

The Essence Store manages metadata associated with each essence in the form of name/value properties. It is possible to add additional name value properties using the SetEssenceMetadata operation. Certain name value properties managed by the Essence Store cannot be set by this method. An Example of read only property is the creation time of an essence. All properties are serialized as strings in the SetEssenceMetadata message.

GetEssenceMetadata

The GetEssenceMetadata operation is used to return the metadata for an essence, a group of essences or all the essences in a silo.

GetEssenceAccessors

If an essence currently has its content stored in a location that provides for http or ftp access then URL's for an essence can be discovered using the GetEssenceAccessors operation.

UpdateJob

This operation can be used to update the priority on a running job.

CancelJob

This operation is used to cancel a currently running job. Jobs can be cancelled whilst they are processing (i.e. doing the work), but if they have reached a post-processing state (finishing the work) they cannot be cancelled, as the work cannot be undone.

GetJobDetails

To retrieve the current state of a job, the state which will be one of: PreProcessing, Processing, PostProcessing, Completed, Cancelled, Failed, and Suspended.

GetEssenceTiers

To determine which storage tier[s] an essence is currently held on. Content can be held in any number of the following tiers: online, nearline or offline.

GetSiloDetails

The GetSiloDetails operation is used to return the information about a specific silo.

GetMediaDetails

This operation is used to return information about which essence content bytes are stored on a particular piece of media (backup tape).

GetLeaseDetails

This operation returns the current status of a lease previously taken out using the MakeAccessible or ImportMakeAccesible operations.

ExtendLease

This operation allows an extension to be made to an existing lease. If a lease is not extended it will eventually timeout.

ExpireLease

This operation should be used if a lease needs to be forcibly expired. This maybe to free space in the online tier used to hold leased essences.

2.1.2 Long Duration Operations

Listed below are the long duration operations currently implemented by the Essence Store.

Import

The import operation is used to copy content into an essence previously created by the CreateEssence operation. If content already exists in the essence then it will be replaced. Essence content is pulled from a URL into the essence store. A CRC code can be passed as a parameter of the operation and the Essence Store will verify that the content retrieved has the same CRC value. More details of the Import operation can be found in section 8.2.2.

Export

The export operation is used to export content from an essence into a user specified location. The Essence Store will copy the essence bytes to the URL location supplied. More details of the export operation can be found in section 8.2.2.

ImportExport

This operation performs an import immediately followed by an export. This is to ensure that the content is first imported and then immediately copied to a location where it can be used in a content workflow.

MakeAccessible

This operation is used to ensure that at least one copy of an essence content is maintained online or nearline for some period of time defined by a Lease. If essence content is maintained online then that content will become accessible for reading via URL.

ImportMakeAccessible

This operation is used to import essence content and then immediately place a lease on the essence to ensure that it remains online or nearline.

2.2 Model

This section describes model underpinning the Essence Store. The model describes the key entities involved in the movement and storage of an essence. The model abstracts away from the specific lower level storage products that the Essence Store actually uses. The implementation is therefore flexible and decoupled from the actual storage product used and could support other types of storage system with minimal additional code. Figure 2 illustrates all the main entities which are defined in the following subsections.

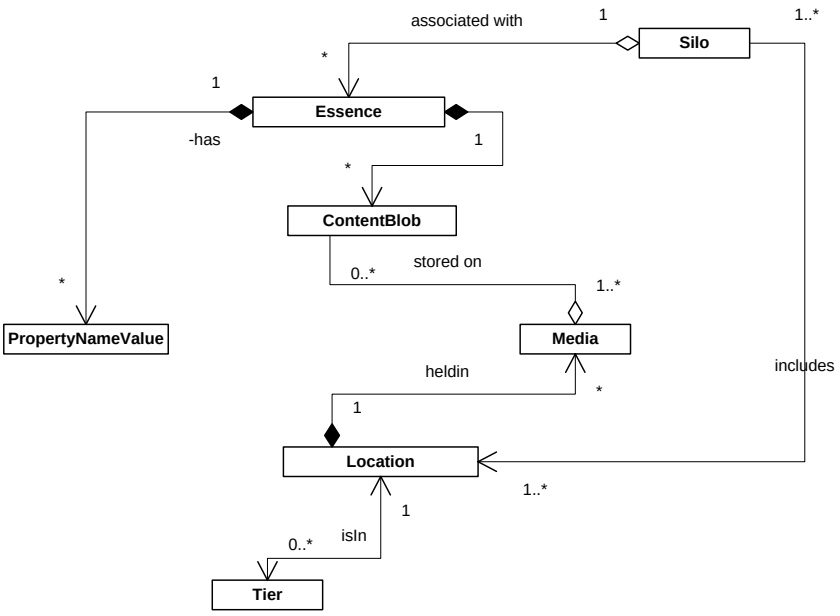

Figure 2: Essence Store Entity Model

2.2.1 Essence

An essence is composed of a content blob and a set of metadata properties. The metadata properties are simple name value properties with a single value.

Essences may vary in size from kilobytes to terabytes in size, and may be stored on disk or on several possible removable media, such as magnetic tapes. Essences may comprise several copies on different media for redundancy and caching reasons. Essences may be exposed and accessed via standard protocols such as http, ftp and rtsp.

A content blob is an immutable array of bytes. When an essence is first created it does not have a content blob. A content blob can only ever belong to a single essence. An essence can only own a single content blob at any one time. The content blob for an essence can be replaced with a new content blob. Old content blobs are not generally maintained by the Essence Store and there is no way of accessing old content blobs (except in certain corner cases involving already open http and ftp connections).

2.2.2 Silo

Essences are segmented into one or more storage silos. A silo is a logical policy container for essences. The purpose of the Silo concept is to enable a solution to be able to segment the essence data into various groupings that can be mapped to storage solutions that have different policies and different characteristics. For example, video and image proxies might be stored on a SAN disk array for immediate access, while much larger master assets might be stored on a HSM system for archival purposes. The silo policy might define that three copies of the essence are maintained at all times and the master essence will most likely remain on an offline tape for most of the time.

Storage Management policies may be associated with a particular silo. The association of storage policy to a silo is HSM implementation dependent. One deployment approach is to assign a single directory to store all the content blobs for a silo, to which a single policy can be associated.

2.2.3 Tiers

The Essence Store models storage tiers roughly according to access latency. It recognises three distinct tiers which are Online, Nearline and Offline. If an essence contains a content blob (new essences are logically zero length) then that content blob may exist on several different storage tiers at once, because there are multiple file copies of the content. If the content blob is online then it is directly accessible via one or more assessor URLs. If the content blob is Nearline or Offline then the content blob will not be immediately accessible. Figure 3 shows some possible transitions between the three tiers.

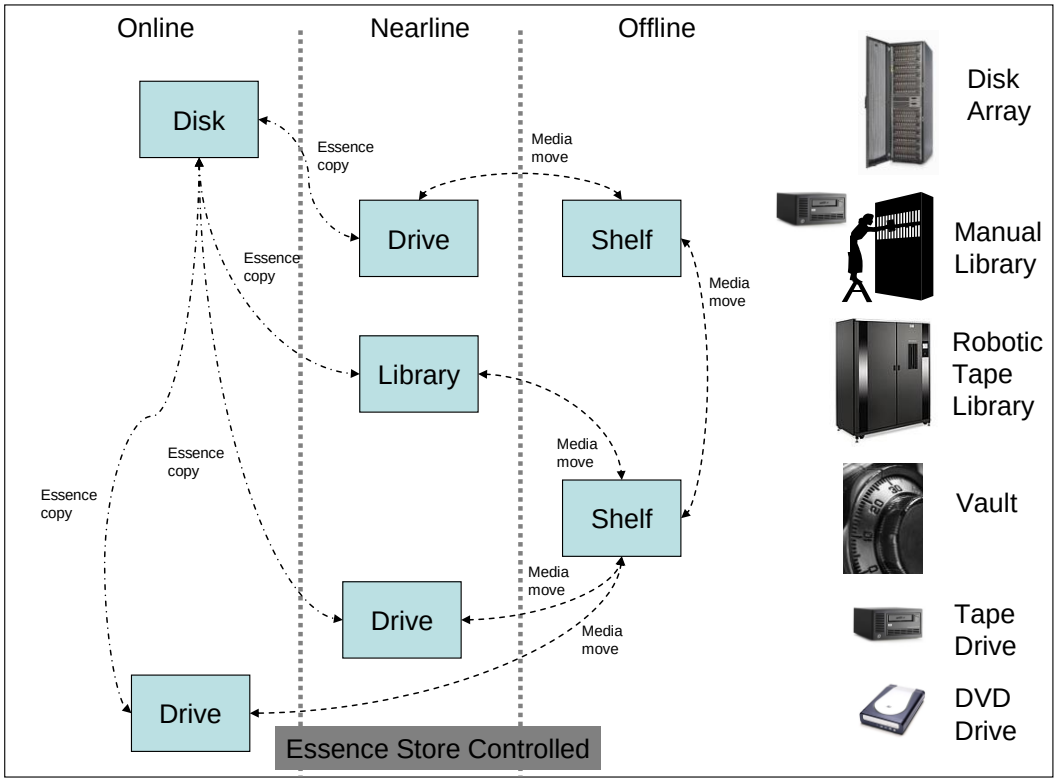

Figure 3: Online, Nearline, Offline

2.2.4 Events

The Essence Store sends out several types of event when certain state changes occur within the Essence Store. Applications may subscribe for these events and the Essence Store will send events directly to the application. A simplified event delivery mechanism was developed that will be easily adaptable to use a WS-Eventing [2] implementation as and when a stable complete implementation is available and can be integrated. All events are sent to an http callback address, if specified by the application.

There are two broad classes of events. These are *fulfilment events* and *lifecycle events*. Fulfilment events are specific to one job and one essence. These events are sent to the application callback address that is (optionally) specified in the request message of all the long running operations. If no callback address is specified then no events will be received. Note that only events related to a specific job and essence will be received; if events are required for all jobs then the callback address must be supplied with each long duration operation that is invoked.

Lifecycle events relate to the lifecycle of various entities such as leases and essences (e.g. essence creation). These events are sent to general subscribers which must be defined in the Essence Store configuration. Filters can be added to control which events are sent out.

The complete list of fulfilment and lifecycle events can be found in the Appendix.

2.2.5 Leases

When using an HSM system to store essence files which are to be used in a workflow, it is advantageous to ensure that a file is kept either online, or at least near-line (i.e. in the robotic library), for the duration of the time that the file may need to be read by the executing workflow. This avoids latency being incurred while waiting for an offline tape, containing the file, to be inserted into the robotic library by a human. The concept of *leasing* an essence provides a mechanism to support this. Furthermore, a *lease* provides a means to pre-emptively move files to a specified tier (online or near-line) ahead of an expected workflow requirement.

An application can take out a lease on any essence contained in an HSM backed silo (i.e. a "StorNext Silo") to request that the associated content file is kept either online, or nearline for the duration of the lease. All leases have a lifecycle and once it has expired, the Essence Store will delete the online file to free storage space or move the tape out of the robotic library.

2.2.6 Jobs

Whenever a long duration operation is invoked a job is created and stored by the Essence Store. Some pre-validation of the message is performed such as validating references to essences or other entities and if any check fails then an error message is returned and the job is not created. If the job is admitted for processing it may not execute immediately, but be held in a queue waiting for resources to become available, or conditions to be met before executing the job. Thus, the Essence Store will continue to accept new jobs even if it cannot process them when they are submitted. The job information is persistent and will survive application server restarts / failures and in a clustered deployment this permits the job to be retried by different cluster members.

All jobs at any time are in one of the states defined in Figure 1.

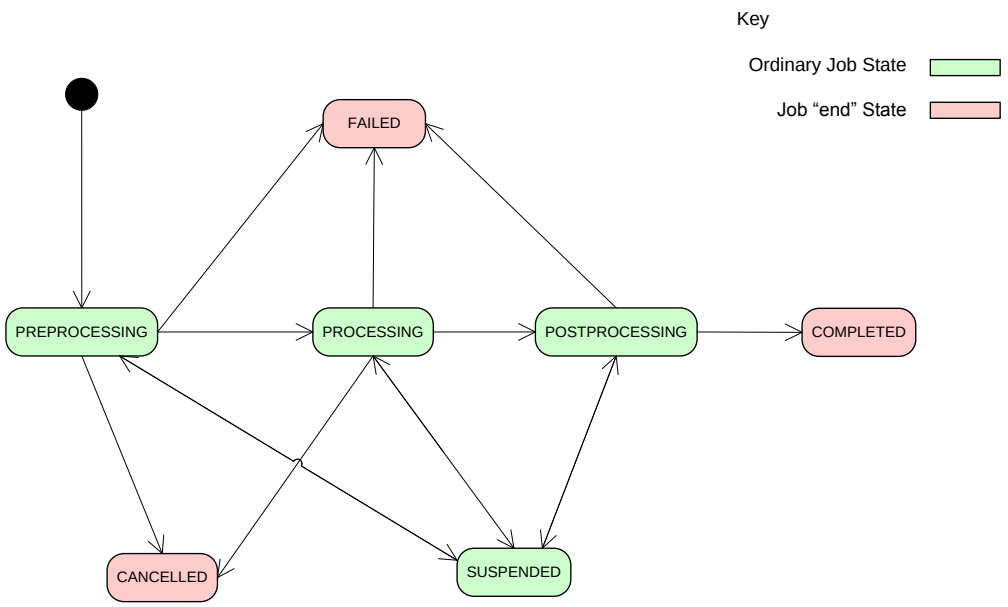

Figure 4: Job State Machine

During successful execution of the job the job state will move from pre-processing (waiting to run) to processing, to post-processing (period to clean up resources) and finally to completed. Any job that is in the prepossessing and processing steps can be cancelled. If a job is an end state then it cannot be cancelled. Cancellation of a job may not occur immediately and can be implementation dependent. For example, during a file import the status of the job must be checked periodically to determine if the import should be abandoned.

2.2.7 Media

Essence data is stored on some storage medium which might be magnetic disk, optical disk, or magnetic tape. An essence file may be copied on to several different media for redundancy reasons. The media can be physically moved from one location to another, the Essence Store can control some aspects of the location of a piece of media, other aspects are delegated to other media control/tracking systems. When essence data in imported the data may well be copied several times on to magnetic tape which may in turn be stored in a separate location. When a request is made to export or access some essence data, the Essence Store may need some media moved from a shelf into a library where it can be read.

3 Implementation View

This section describes the internal design of the Essence Store service.

3.1 Architecture Overview

Figure 5 shows the major structural components that have been developed. The application tier (top yellow block) interacts with the Essence Store at several points. The Essence Store web service interface is the main contact point, providing operations to query the model state, start and manage essence jobs (such as import or export). The secondary interface points include FTP and HTTP access points provided by the Essence Store. These support ftp and http accessors, and may be optionally deployed. Typically a staging FTP server is provided by default. There may also be shared file system resource accessible by both the Essence Store and the application. The Data Mover component is able to read to or write from essence files to external http or ftp servers. It also supports internal file transfers. Each of the principal Essence Store components also contains instances of the DiskOnly Silo and HSM Silo implementations that read and write to a specified storage type. In the case of the HSM silo, an agent, in this case a StorNext agent, is also required as part of the HSM Silo implementation. The StorNext agent interacts directly with the StorNext controller.

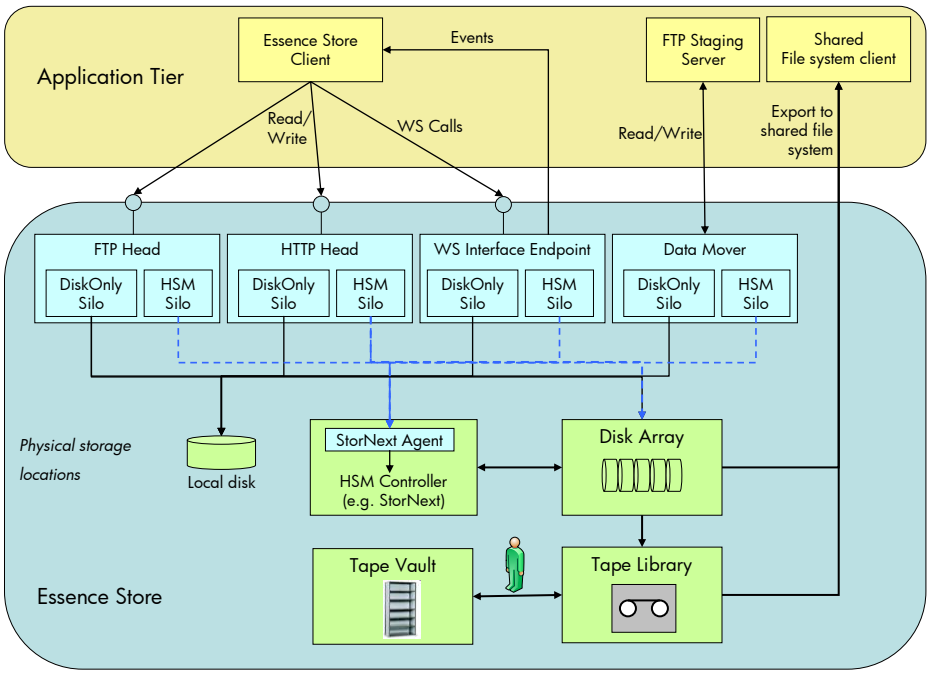

Figure 5: Essence Store Implementation Architecture

The Essence Store service instance is deployed as a set of service components (blue) and infrastructure components (green**)**. The green components provide the storage infrastructure over which the Essence Store operates.

The web service interaction model between the solution and the Essence Store service is complex and being both bi-directional and asynchronous. Calls made from the solution to the service often include one or more asynchronous call-back events. The solution may also interact with the Essence

Store by way of ftp and http access via special ftp and http code that is hardened to handle 100GB file access. The Essence Store can be deployed on one or more machines, referred to as *heads*, to provide scalability and fault tolerance. A *head* is a deployment of some Essence Store code onto a server. A complete Essence Store deployment typically includes many servers and potentially a heterogeneous mix of components running on each head. A single developer deployment is provided which deploys all components to a single head.

3.2 Component Model

The Essence Store high level design is centred on the concept of components that work together to implement the required functionality. There are many types of component within the Essence Store design each encompassing a specific functional area. Components are large scale granular software composition structures and not low level object reuse classes. Components have the following characteristics:

- They have an interface and a factory.
- They implement the IComponent interface.
- They are constructed at head initialization time.
- They are not dynamically created or destroyed after service initialization.
- They implement a set of methods specific to their functionality.
- They have their own section in the configuration data model which they use.
- They expose themselves as a JMX object.
- They may depend on other local or remote components (local is within the same head).
- They are independently Testable with use of other mock components.
- Every component instance has a name which is unique within a head.
- Components with the same name (on different heads) are equivalent.

Figure 6 shows the principal components that exists in the Essence Store.

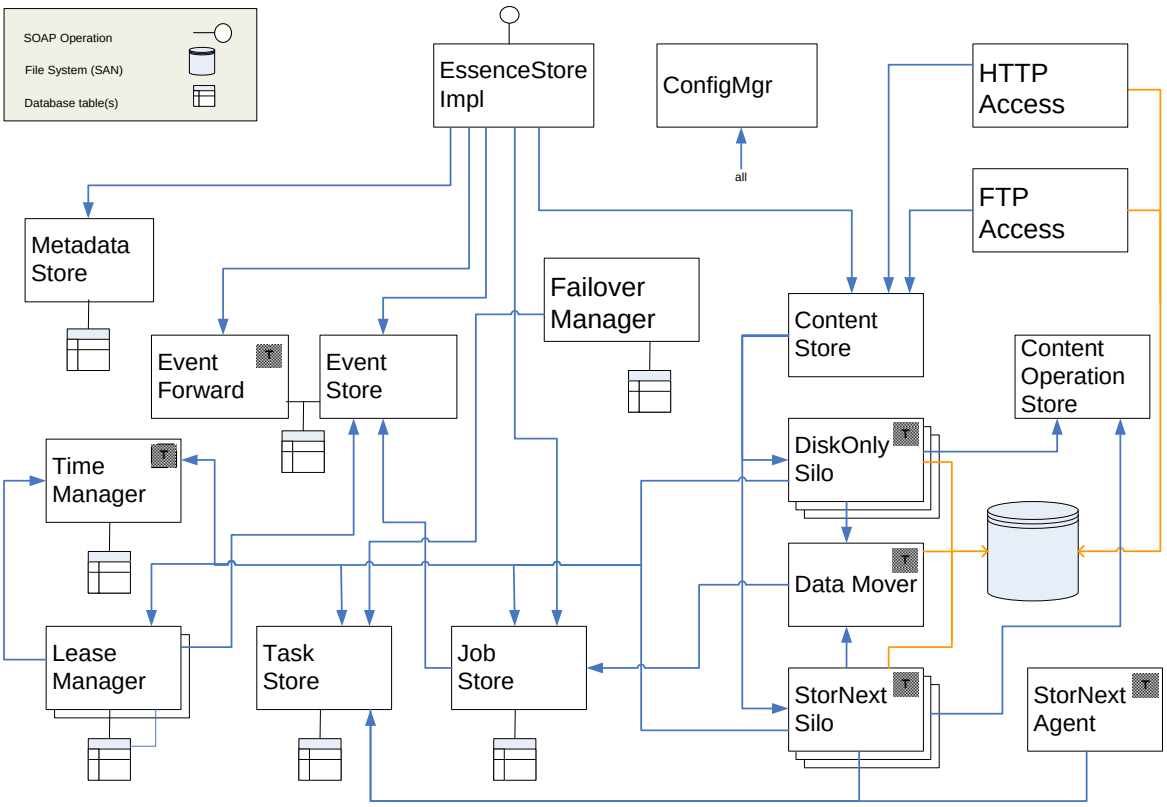

Figure 6: The principal components of the Essence Store

3.3 Summary description of the components

This section contains a short description of each component. Some of the components are described in more detail in later sections.

3.3.1 ConfigMgr

The ConfigManager provides an interface that will allow any object in the system that implements the Configurable interface to ask for its configuration details, (all components are configurable). The ConfigManager will pass back the relevant plain old java object (POJO) containing the configuration details.

3.3.2 ContentStore

The ContentStore Component mediates all content blob manipulation. This includes reading and writing content from a Silo, deleting content, and killing currently executing read and writes tasks. The ContentStore delegates all content blob operations to a component that implements the *Silo Interface*. The ContentStore may be configured to use several types of Silo implementation and there may be several Silos configured for each deployment.

3.3.3 DiskOnly Silo Implementation

The DiskOnly silo implementation implements the silo interface over a standard file system.

3.3.4 StorNext Silo Implementation

The StorNext implementation implements the silo interface for a managed StorNext file system.

3.3.5 ContentOperationStore

The ContentOperationStore is used to manage the many-to-many relationship between Jobs and Tasks. This is done by maintaining a link table between Jobs and Tasks.

3.3.6 MetadataStore

The purpose of the MetadataStore is to hold both Essence Store metadata, defined to be any metadata created by the Essence Store and application metadata that is metadata provided by the application using the Essence Store. Metadata consists of name value properties. Use of the MetadataStore is generally discouraged for application metadata, which should be ideally stored in a separate system. However, it is sometimes helpful to be able to associate an application identifier with the essence managed by the Essence Store service.

3.3.7 DataMover

The DataMover component contains numerous threads for performing files copies from one location or tier to another. Each file copy is managed as a Essence Store Task. There are a variety of different source and destination locations that the DataMover supports. Once the bytes are copied, the DataMover will update the task status to indicate successful completion.

3.3.8 JobManager

The JobManager stores details of every job and sends out events (via the EventStore) at important stages in the lifecycle of a job. The job state is persisted in a relational database.

3.3.9 TaskStore

Tasks are low level atomic, but long running work items. Each task has an associated task type. The TaskStore manages each task. There are several different types of tasks and they are processed by dedicated components. The task state is persisted in a relational database. Each task can have arbitrary name value properties belonging to it. New task types can be defined and new components created that are able to process those task types.

3.3.10 EssenceStoreImpl

The EssenceStoreImpl is responsible for dispatching incoming web service requests to the appropriate component(s) and responding to the original request. Additionally, the EssenceStoreImpl dispatches internal asynchronous responses (also called re-entrant requests) driven from completed tasks for previous jobs currently being executed. For each in-bound request, either new or re-entrant, that is part of a long duration job, the EssenceStoreImpl component will dispatch the request to the appropriate state machine that encapsulates the logic required to fulfil the request.

3.3.11 StorNext Agent

This is both a component and a Restful web service. The component always runs on the StorNext controller machine. This is because the StorNext API could only be accessed from the controller

node and not from a node with the just the StorNext file system mounted. The StorNext Agent accepts both web requests and picks up tasks from the TaskStore. Asynchronous requests are picked up as tasks and results are persisted in the same task. The task state is updated upon completion and will then be processed by other components. Synchronous requests are sent to the StorNext agent by Restful style web service calls. This helps to minimise the complexity of deployment of the StorNext agent.

3.3.12 EventStore

All Essence Store components that generate events first persist them using the EventStore. The EventStore is responsible for persisting events to a relational database which can then be sent to the registered callback address by the EventForward component.

3.3.13 EventForward

The EventForward component is responsible for delivering the events managed by the EventStore to a callback address either configured as a subscription or sent in an initial long duration request. The EventForward component guarantees messages are sent at least once. Duplicate message delivery is possible, but the application can easily handle this eventuality by inspecting the event identifiers and either the associated job, essence or lease identifiers. Events are guaranteed to be delivered in-order.

3.3.14 Failover Manager

The Failover Manager component is responsible for detecting when another head has failed and cleaning up any blocked tasks owned by components running on that head. Each Essence Store head has a randomly generated Identifier HeadID, with an associated last update time that is used to indicate whether a head is still active and able to connect to the database. The operation of the Failover Manager is tightly linked to the task model and is described in more detail in section 3.4.1. The Failover Manager component must be deployed on all Essence Store heads.

3.3.15 FTP Endpoint

This component is a FTP server that provides remote access to online essence content.

3.3.16 HTTP Endpoint

This component is a HTTP server that provides remote access to online essence content.

3.3.17 Lease Manager

The Lease Manager stores all the details of every lease. It manages all leases providing functionality to create, update and deleted leases. The Lease Manager is also responsible for generating lease lifecycle events at the various important stages in the lifecycle of a lease.

3.4 Task and Job Mapping

The Essence Store web service interface exposes several types of operation, some initiate new jobs others retrieve information, others provide administrative operations. Jobs are first class entities in the Essence Store interface. A job has an associated type which is tied to the operation that generates instances of that type of job. A job is generally broken down in to a set of simpler tasks

that are performed sequentially. Each job type has its own logic implemented as a state machine that sequences the execution of the tasks that make up the job. Both Jobs and Tasks are persistent entities and they will survive server restarts. If a job is cancelled then all in-progress tasks for that job will be killed, although there may be a short delay before that occurs. Task details are stored by the TaskStore component. Tasks have a type property that stores the type of task that needs to be performed. Tasks may also have any number of name-value properties that can be used to transmit parameters between the task creator and executor. Tasks are executed by components; the task type must match the components valid task types for a component to execute a task. When a component executes a task it first *claims* the task to prevent other component instances from also processing that task. This is done by setting the headID and component name values into the properties in the task. The setting of the headID values is done in a database transaction in such a way that no other component could have claimed the task before or after the transaction.

The claiming process is used by all components that process tasks. Each component that can claim a task and execute it has a thread pool. The threads in the pool periodically attempt to claim any unclaimed tasks (of the correct type). The size of the thread pool for a component constrains how many tasks can be executed concurrently. Tasks can be marked as blocked on another task, marked as blocked for some time period or marked as awaiting some external stimulus (e.g. a tape being moved from a vault to a library).

Upon completion of various tasks, certain fulfilment events may be generated (e.g. DataDownloadComplete) to provide the application indications of progress.

3.4.1 Failover and the Failover Manager

Each Essence Store head has a Failover Manager component, that works with the other Failover Manager components (deployed in other Essence Store heads) to ensure that when a head fails that tasks are retried and that there are no blocked tasks. Each Failover Manager maintains a row in a database table that contains the headID of each active head as primary key. Each Failover Manager periodically (e.g. every 30secs) wakes up and updates its headID row with the current time. Any time drift between Essence Store heads is compensated for. When a component in a head claims a task, it marks the task with the headID of that head. If for any reason that head fails or cannot communicate with the database, the Failover Manager will not update its row. The other Failover Managers will after updating their row, check that their peer Failover Managers have updated their rows recently. If a Failover Manager has not updated its row within some (configurable) time then it will be considered to have failed and the other Failover Managers will proceed to unclaim any and all tasks that had were claimed by components in the failed head. The unclaimed tasks will become immediately available for claiming by other components in other heads. If a head is alive, but is part of a network partition that does not include the database then it will have its tasks unclaimed by a Failover Manager that is part of the network partition with the database. A component must check at relevant times (and always before finishing a task) that the task has not been unclaimed by a Failover Manager during the execution of task. In such cases the component must stop all work on the task and clean up. It is highly unlikely that a component would be able to proceed whilst the Failover Manager is unable to update the database table.

The combination of the task claming model and the (redundant) deployment of many Essence Store heads with Failover Managers provides a design that can both scale and handle many types of failure.

3.5 Configuration and Essence Store Deployment

The Essence Store design supports the deployment of heterogeneous heads where the types of component in each head can be configured along with specific configuration values for each component. For example multiple http and ftp heads could be provisioned to support higher IO requirements. Figure 7 shows an example Essence Store deployment consisting of three identical servers running Essence Store interface endpoints and http and ftp endpoints. WebLogic Application Server 9.2 (running on Red Hat Enterprise Linux 4 AS) was used as the hosting environment. The Jetty HTTP server [5] is deployed on the primary and secondary StorNext metadata controller hosts containing the StorNext Agent and TaskStore components.

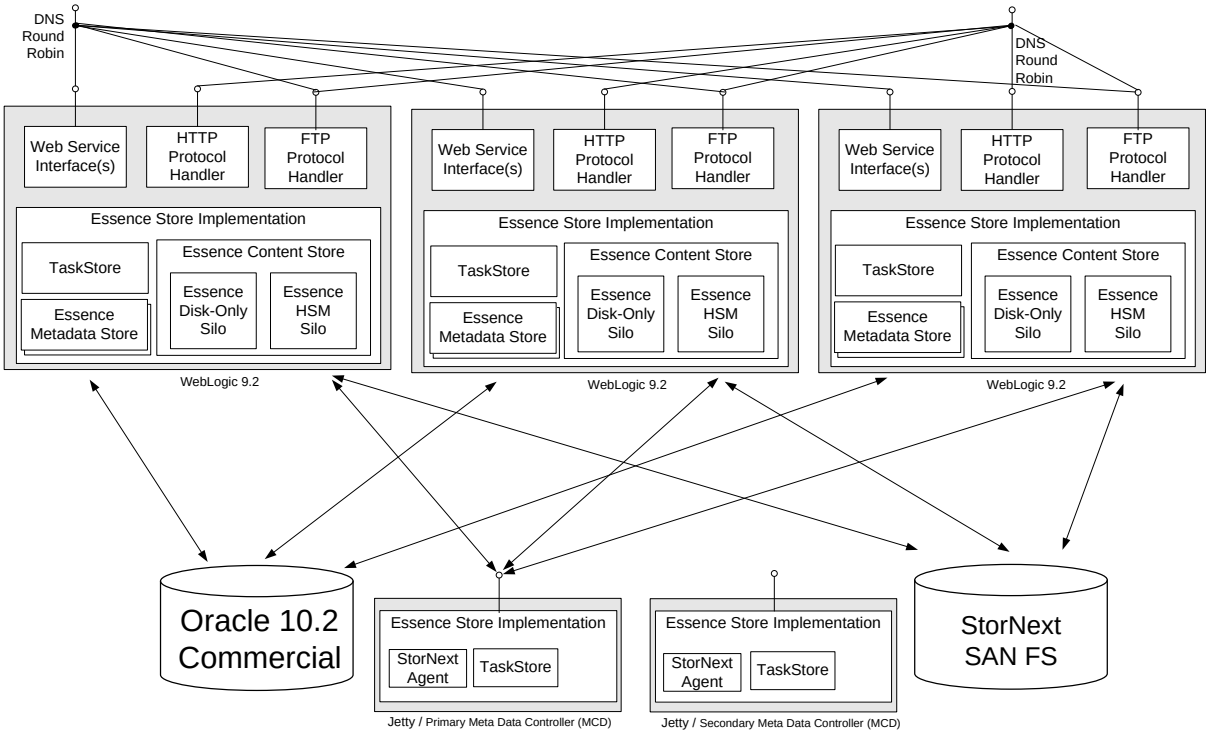

Figure 7: Example Production Configuration of the Essence Store

In order to use the Essence Store in conjunction with a StorNext silo, the StorNext Agent must be deployed on the StorNext Metadata Controller (MDC) as it must be able to access the StorNext C API via JNI. The StorNext Agent running inside Jetty, receives restful requests from the main Essence Store service for synchronous requests and claims tasks from the task store and calls the StorNext API in order to execute the request or the task.

To provide load balancing in front of the Essence Store, Linux Heartbeat has been used. This was to avoid placing a single load balancer between the Essence Store heads and the import and export locations, which could become an IO bottleneck. This heartbeat deployment is illustrated in Figure 8

. The WebLogic application servers are configured to listen on a wildcard IP address. All servers have the Linux Heartbeat program installed.

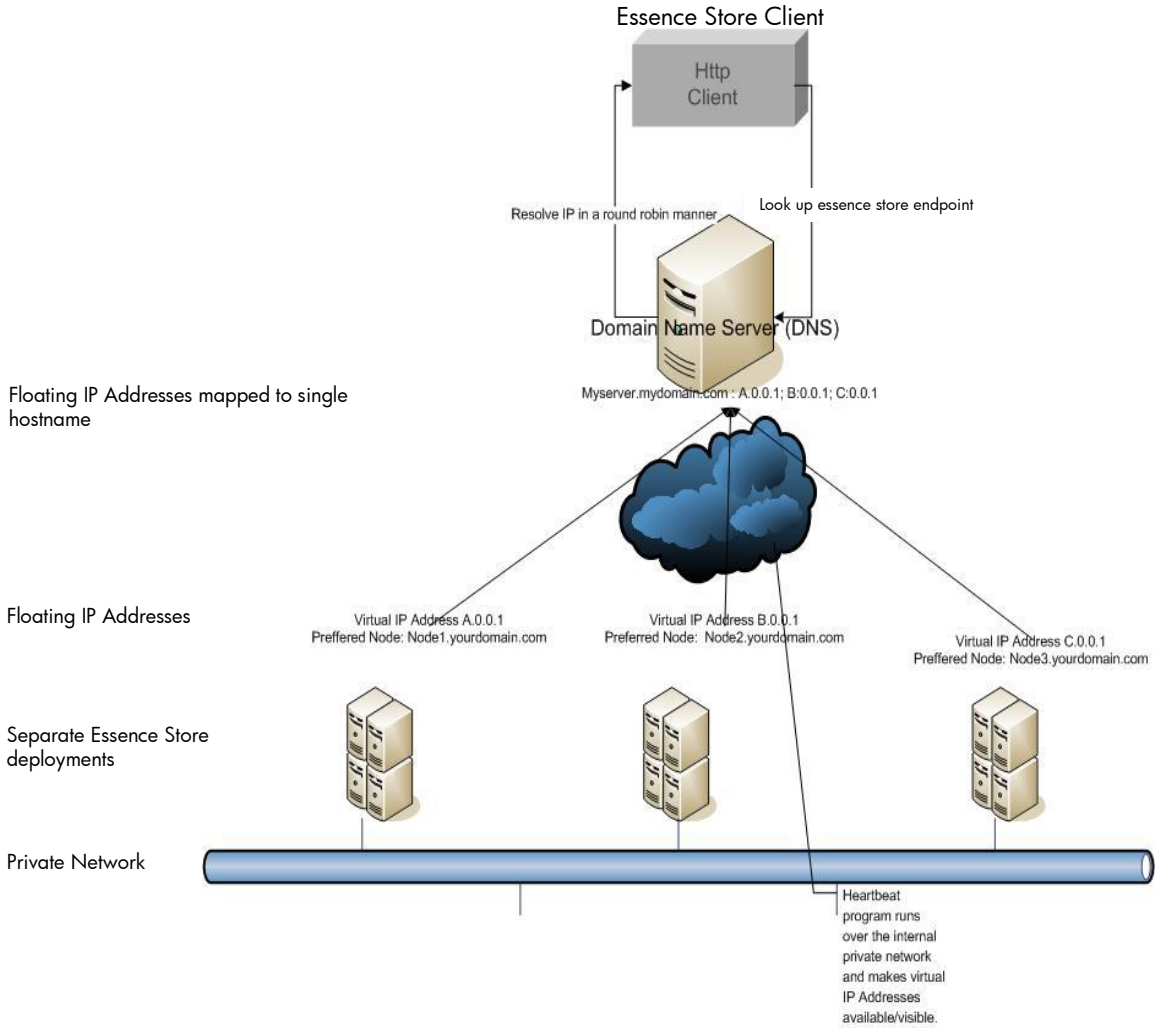

Figure 8: Using Linux Heartbeat for DNS Round robin load balancing

The Linux Heartbeat program is responsible for self organization of resources amongst servers available to run these resources. Resources in this case are *floating* IP addresses that must be made available on each of these servers. To avoid contention between the OS and the Heartbeat program for IP addresses, the floating IP addresses configured to be made available by Heartbeat should not have been assigned anywhere else in the private network. Furthermore, the floating IP addresses are configured to be mapped to a single DNS alias. When a client resolves the alias, the IP addresses are returned in a round-robin manner ensuring the same IP is not chosen each time thus providing load balancing.

For failover, when one of the servers crashes, or loses network connectivity, the Heartbeat daemon running on the other two nodes (servers) detects this by multicast and broadcast communication, ensuring that one of the nodes claims the floating IP address of the failed node. As the WebLogic application server is listening on a wild card IP address, it should continue to be able to serve

requests via the newly claimed IP. In this configuration, if auto-failback mode is enabled for Heartbeat, i.e. when the lost server comes back into the cluster, the IP will be re-claimed by its original i.e. preferred node.

In order for an Essence Store client to be hidden from the details of virtual IP addresses and failover from one server to another, it is recommended that a DNS alias is set up and the floating IP addresses in consideration are mapped to a DNS alias.

3.5.1 Scalability

Total ingest rate is a very important customer metric for the Essence Store. With sustained write rates in the 15 Terabyte a day range, the architecture of the Essence Store needs to be able to scale at several levels. The Essence Store can have many web service heads using DSN round robin to provide load sharing. File read and writes from/to file systems and URLs is primarily performed by Data Mover components. Essence Store heads containing just Data Movers can be deployed to provide extra bandwidth as necessary. The task model described in section 3.4 supports task level scalability only limited by the processing power of the server cluster and network bandwidths. Several robotic cabinets, each with several tape drives, can be installed to increase the aggregate bandwidth for writing to tape. The SAN attached disk capacity needs to be sufficiently large to be able to cache both the incoming and outgoing files.

3.5.2 Configuration

One of the major challenges of the Essence Store was to be able to minimise the complexity of configuration as much as possible. Given the potential numbers of components and Essence Store heads in a production environment configuration could become error prone.

To help simplify configuration, a model of using a single point of configuration was adopted. The configuration was expressed in an XML format that could contain configuration for several named heads while allowing for global (default) values to be specified and inherited by each head.

During start-up, an Essence Store head initializes its local ConfigurationManager component. The ConfigurationManager searches in three locations for the configuration. This is illustrated in Figure 9. The first location is a database table, the second location is an external XML file specified by a system property, and the third and final location is an XML file contained in the Essence Store jar file. This latter option is convenient for providing a default configuration for a developer deployment.

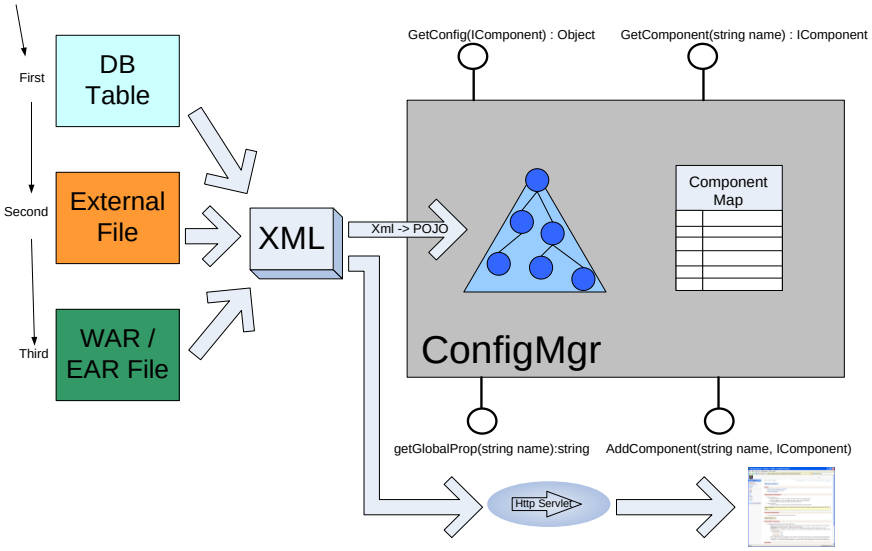

Figure 9: Configuration Search Order

One difficulty with using a database table as the source of configuration information, is how does a head know its database configuration before reading its configuration. In a J2EE deployment this is possible by configuring a DataSource object which contains the database connection details. This is convenient because WebLogic 9.2 provides an administrative user interface to configure DataSources and also provides with additional testing facilities that do not have to be provided by the Essence Store implementation.

Wherever the configuration data is found it always has a single form, which is an XML document. Each component has its own XML element that holds its configuration. JAXB [6] is used to convert from the XML schema to set of POJOs that the component can used to read its configuration.

During start-up, a component requests its configuration from the local Configuration Manager using the `GetConfig` method. The Configuration Manager determines the class type of the component which it matches to the appropriate configuration schema element type. In addition the Configuration Manager determines the local hostname which allows the Configuration Manager to uniquely identify the XML configuration fragment for the requesting component running on the current server host. If a configuration fragment is not defined for the component running on the local host then a default configuration fragment is used instead.

3.5.3 Diagnostics

The Essence Store provides a simple diagnostics pages to assist in initial setup. Every component produces diagnostic output that is rendered as HTML in the diagnostic page. Each Essence Store head serves a single diagnostic page that combines the output from all components deployed in that head. An screen shot of a diagnostics page is shown in Figure 10.

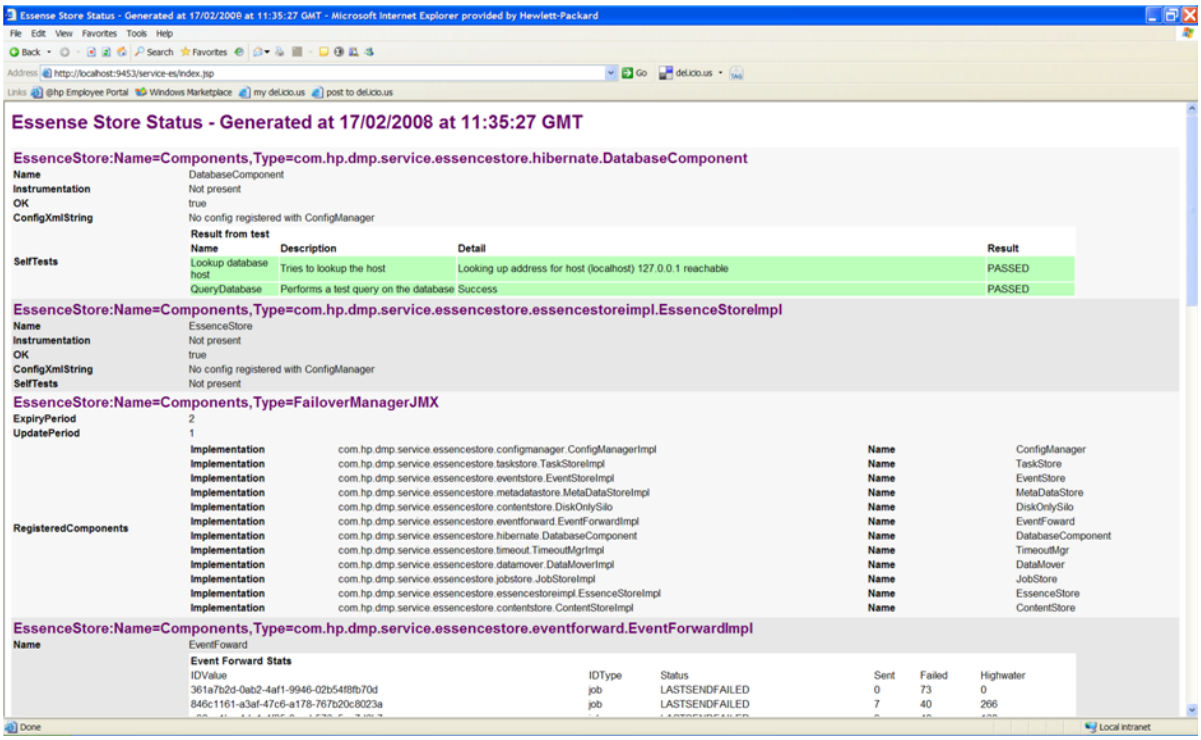

Figure 10: Example Screenshot of the Essence Store Diagnostic Page

The diagnostics page includes a list of components running inside that Essence Store head. Within each component section, the following information fields are displayed:

- Name of the Component.
- Instrumentation - Not all components will provide instrumentation; however those that do will include a data count of various data that has been processed or is processing in the particular component.
- ConfigXMLString - The snippet of the XML that the Component has been configured with.
- SelfTests (a set of self tests) - Several of the components perform quick self tests to validate whether they are setup correctly.
- Report – Provides a small snapshot view of the current state of a particular component or the tasks/jobs completed by the component.

4 Tape Management

An important operational aspect of the Essence Store is the management of tape media both inside the robotic library and offline in the digital vaults. It's important that tapes are efficiently moved between library and vault in order to support the essence file movement. This is a critical function performed by the operations team.

The Essence Store service interface provides a GetMediaDetails operation that can contain an essence, media or silo identifier as a request argument. The response message contains detailed information describing the media describing its location, utilization, access time and essences stored on the media. The results can be paginated because the response message can be very large. The response can be analysed to identify which tiers an essence is currently held on and to plan which media to move into and out of the library to support incoming work orders. The response message must be analysed carefully because essences file can be held on multiple tapes and may span tapes if very large.

5 Acknowledgements

The Essence Store design implementation testing and deployment was successfully realised due to the dedication of many people, both inside and outside of HP. The authors would like to acknowledge the contributions of the following HP people: John Wood, Richard Lucas, Andrew Nelson, Andrew Morgan, David Wilcox, Stephen Bailey, James Brook, and Sunil P Suthar.

6 Summary

This report summarises the design and implementation of an Essence Store. The Essence Store is a storage service tailored to the needs of the post production studio environment, but could be deployed in another application area where large file management represents a significant challenge.

7 References

[1] Ascent Media Group: http://www.ascentmedia.com/

[2] WS-Eventing: http://www.w3.org/Submission/WS-Eventing/

[3] WS-Security: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[4] Linux Heartbeat: http://www.linux-ha.org/LearningAboutHeartbeat

[5] Jetty Web Server: http://www.mortbay.org/jetty/

[6] JAXB: https://jaxb.dev.java.net/

[7] Hibernate: http://www.hibernate.org/

8 Appendix

8.1 Essence Leases

Applications can request leases for essences by calling the MakeAccessible operation. The MakeAccessible operation makes the content associated with an essence available on the tier specified in the <LeaseDetail> element (either nearline or online). There is a related operation called ImportMakeAccessible which simply precedes a MakeAccessible operation with an Import, to allow processing of the file immediately after ingestion.

A lease can exist in one of six possible states throughout its life. The lease state diagram is shown in Figure 11. The State Diagram below details the relationship between lease states. A change to a lease state can be made *directly* by revoking a lease, or *indirectly* by extending its duration or because the essence is deleted or modified.

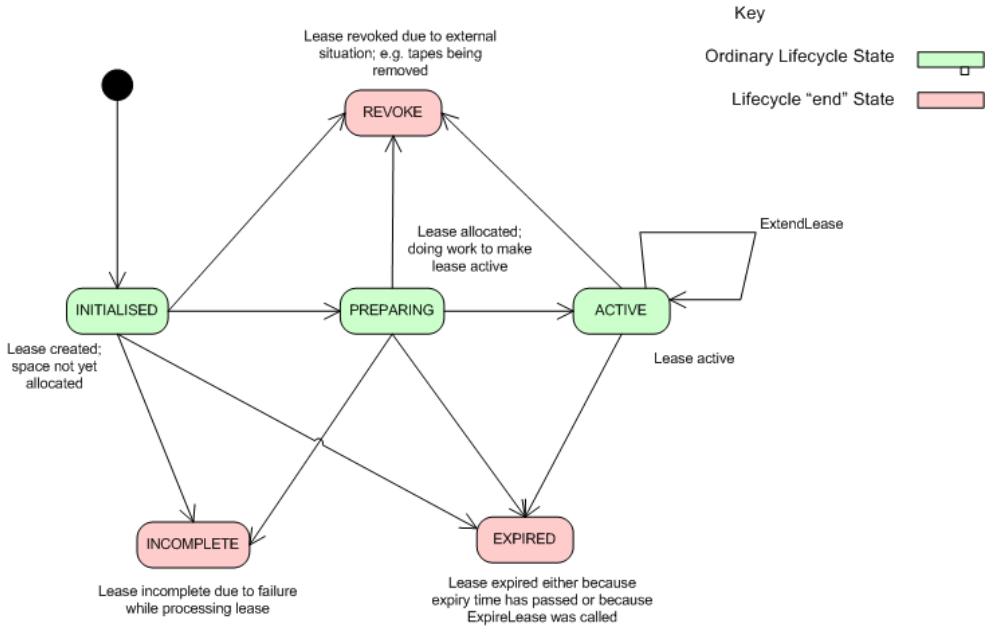

Figure 11: Lease State Diagram

- **Initialised**

Upon creation, a Lease will be assigned the Initialised state. An online lease will remain in the Initialised state until the Essence Store obtains confirmation that there is sufficient space available in the online disk area for a leased copy of the essence to be made. Similarly, a nearline lease requires that there is a tape slot available in the robotic library for a tape containing an essence to be inserted. In some cases, the tape may already be in the library.

From the Initialised state, a Lease can go to Preparing, Revoked, or Incomplete states.

- **Preparing**

A Lease in the Preparing state is waiting to be made Active. The most likely reason that a Lease might spend a significant time in Preparing state is when the essence against which the Lease is

being requested is Offline and is awaiting human intervention to load it into a Nearline (or subsequent Online) Tier.

From the Preparing state, a lease can go to the Incomplete, Expired, Revoked or Active states.

- **Active**

A lease in the Active state will ensure that an associated essence copy is kept on the requested tier. The Lease will remain in the Active state for the period of time specified in the Duration included as part of the MakeAccessible request unless the underlying Essence is deleted, the lease is extended, or the lease is expired.

From the Active state, a Lease can go to the Revoked, Expired or Active states.

- **Incomplete**

A lease will be moved to the Incomplete state if an error, or other failure occurs, while the lease is in either the Initialised or Preparing states that will make it impossible for the lease to be made Active. The Incomplete state is an end state meaning that no further changes in the lease lifecycle are possible. For example, if the Essence Store Job that manages the transition of a lease through the Initialised and Preparing states is killed, the Essence Store will move the lease to the Incomplete state.

- **Revoked**

A lease will be moved to the Revoked state if an event beyond the control of the Essence Store causes an Initialised, Preparing or Active lease to become irrelevant or not-meaningful. The most likely cause of an online lease to be revoked is if the leased essence copy to which the lease refers is deleted. The most likely cause of a Nearline lease to be revoked is if the tape(s) maintaining the essence are moved out of the tape cabinet. The Revoked state is an end state meaning that no further changes in the lease lifecycle are possible.

- **Expired**

A lease will be moved to the expired end state once the period of time defined in the lease duration attribute has passed or if the ExpireLease operation is called.

8.1.1 Lease Synchronization and Lease Revocation

Although the Essence Store creates and manages the essence leases it cannot guarantee that the terms of the lease during the time that the lease is active are met because users of the system can delete files that have been moved online and/or remove tapes from the robotic cabinet that have been leased nearline. Ideally, operators should *always* refer to the active leases before undertaking any such actions. In addition, during system downtime a lease expiry time may be passed, which the Lease Manager cannot update because it is not running. In both cases when the system is next running, the Lease Manager will endeavour to synchronize the currently active leases with the real state of the system. If a user initiated changed outside of the Essence Store means that the conditions of a lease are no longer met then a lease revoked event will be sent out.

8.1.2 Essence File movements related to leases

Figure 12 illustrates the file system layout and the file/tape movements that are caused when leases are taken out or removed from essences. An important aspect of system configuration to support leasing is to provide a directory on the StorNext file system reserved for leased files. This is denoted "Online leased content" in Figure 12. This lease directory does not have a configured storage policy and so management of the files is solely under the control of the Lease Manager. This is important to avoid the automatic truncation of files. The configuration file can be used to specify how much storage is allocated to hold the contents of the lease directory. This is important to avoid the leased area from taking away too much capacity that is required for the silo directories.

When files are imported, they are written to a directory corresponding to a silo. According to the storage policy for that silo, the files will eventually be copied to tape and then truncated.

If an essence is leased to an online tier, then if the file is still in the silo directory, it can be copied to the online area. Otherwise the StorNext export operation is called to export a copy of the file to the online leased directory. Note if the tapes containing the file are offline, then a request to the operator will be made to *eject* the tape from offline and *move* it into the robotic library. Eject and move are the primitive operations supported by StorNext that are needed to move a file from offline to nearline.

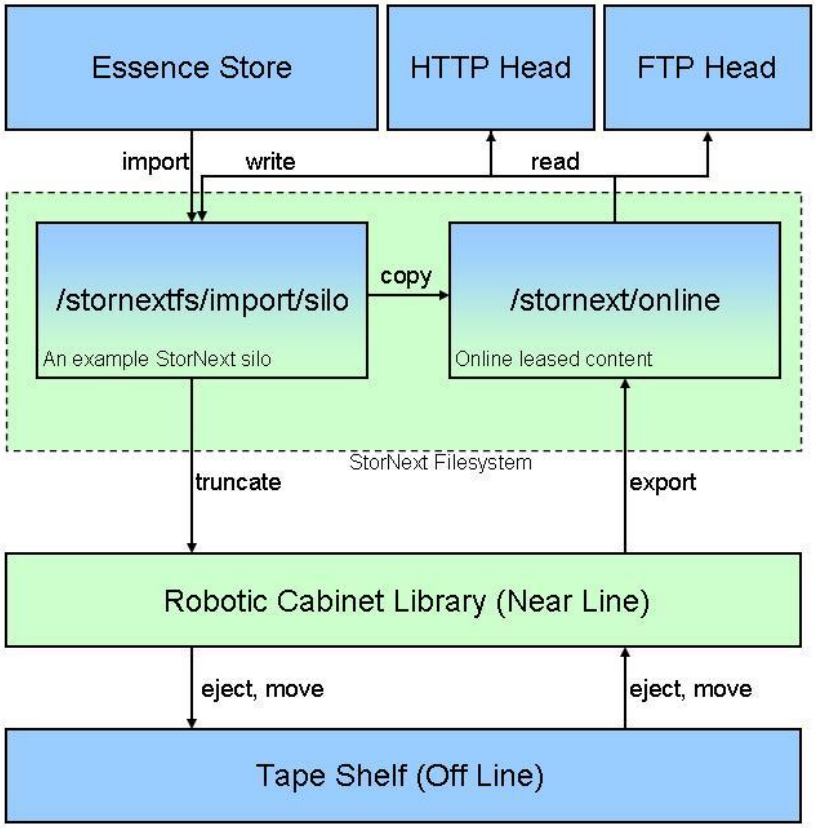

Figure 12: The Online and Nearline Tiers and an Example Silo

When a lease expires, or is revoked, the associated essence file in the online area will be deleted. Note that an essence may have several different leases at any given time and only when the last lease expires will the storage space be reclaimed. Other leases in the Initialised state waiting for storage space, might then be able to be satisfied. During restart the Lease Manager will check the files in the lease directory with the recorded and still active leases and check that they are consistent. Any files that do not have any open active leases are removed.

If an essence is made accessible on the nearline tier, then the primary tape that the content is stored on is ejected from the offline shelf and moved into the nearline robotic cabinet. Whilst the primary tape copy is always used first by StorNext when reading a content file, if the primary tape is unavailable or unreadable then either the secondary or tertiary tapes may be used instead. However, since the tertiary copy is usually held offsite, the Essence Store ensures that the secondary copy is always used if the primary tape copy is not usable.

8.2 Reading/Writing Content into the Essence Store

There are two mechanisms for reading and writing content into the Essence Store. The first approach is to use an Accessor URL. The second approach is to use the import operation, which is recommended, especially for large files.

8.2.1 Ingesting an Essence using a URL Accessor

The Essence Store can be configured to make content available through several standard web protocols such as http and ftp. In order to provide access to the essence content over these protocols the essence file must be first moved onto an online storage disk area. As online disk space is relatively scarce resource it must be managed efficiently. An application must first request that an essence content blob be accessible directly from an online tier. The space required to service the request is allocated and reserved under the leasing scheme. When the lease expires the space will be reclaimed, and the content will no longer be accessible from that tier. Accessors are in this way time limited.

The actual accessor protocols that can be used depend on the particular Essence Store deployment and configuration, the tiers on which the copies of the essence are located and whether the accessor is for reading or writing. The available protocols can be determined at runtime using the getAccessors<EID,read|write> operation on the Essence Store interface as illustrated in step 1 of Figure 13. An essenceAccessors<*URI> message will be returned which will list all the accessor URLs. The returned accessor URLs are dynamically computed according to generation rules in the Essence Store configuration. The application can choose the preferred accessor and can then begin to read or write content using the URL.

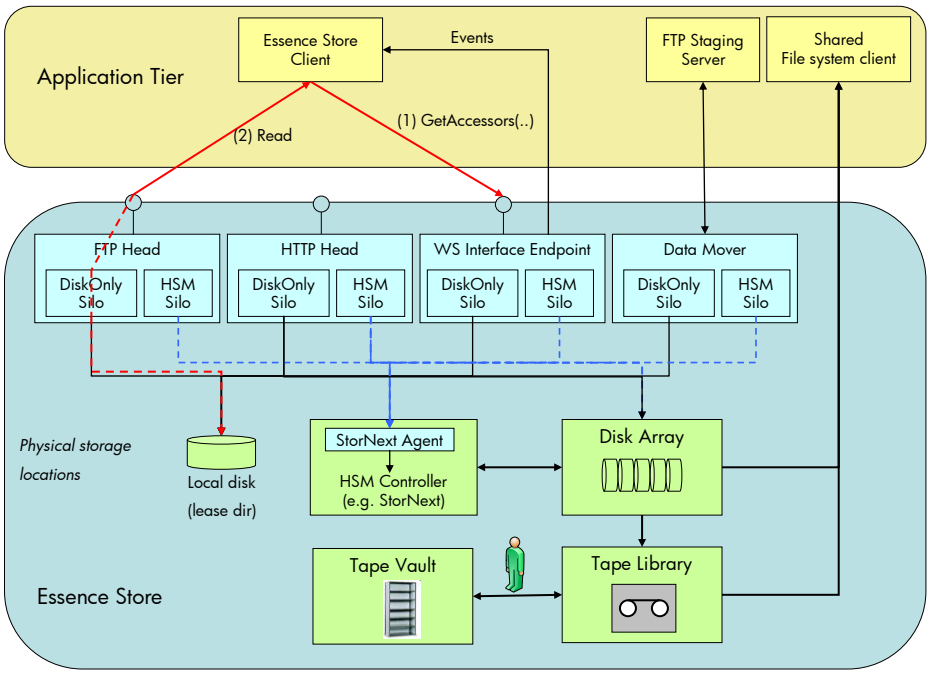

Figure 13: Accessing an Essence using Ftp

Essence content can be modified via an accessor URL if the protocol supports writing. The application will write the new essence content to the URL which will directly update the essence content in the Essence Store. The old content blob will be replaced with the new content blob when the last byte is successfully written to the new content blob. Existing protocol readers of the old content blob will continue to be able to read the old bytes from existing open connections.

Using an accessor to read or write content requires that the application takes on the task of handling retries in the event of failure. However, URL access is convenient for small files and for building web applications.

8.2.2 Accessing an Essence Content Using Import and Export

Instead of writing a content blob to the Essence Store, applications can direct the Essence Store to read the essence content from an external source such as an ftp or http server. This is referred to as a pull model. Import operations requires a pre-existing essence in which to import the content. Strictly speaking, the import operation does not overwrite the existing content blob, it creates a new file for the new content blob and updates the reference to the content blob held by the essence.

There are several advantages for an application to use the import operation rather than pushing data from the solution using http or ftp accessors. These are:

1. The Essence Store can determine the optimum number of requests to execute in parallel.
2. The Essence Store can ensure that the online disk pool is never exhausted by to many large essences being added at once (which can occur with ftp/http push).

3. The Essence Store will perform the import in a fail over manner (another head continuing from where the failed head had stopped).

4. Priorities can be set on imports so that high priority jobs can overtake lower priority jobs in the queue.

5. The Essence Store provides programmatic access to the in progress jobs, allowing for status monitoring and ability to kill jobs.

6. The Essence Store could potentially use more than one reading thread to speed up import.

The steps in an import operation flow can be seen in Figure 14. The export operation follows a similar pattern expect that the export may be to an FTP location or a shared file system client directory.

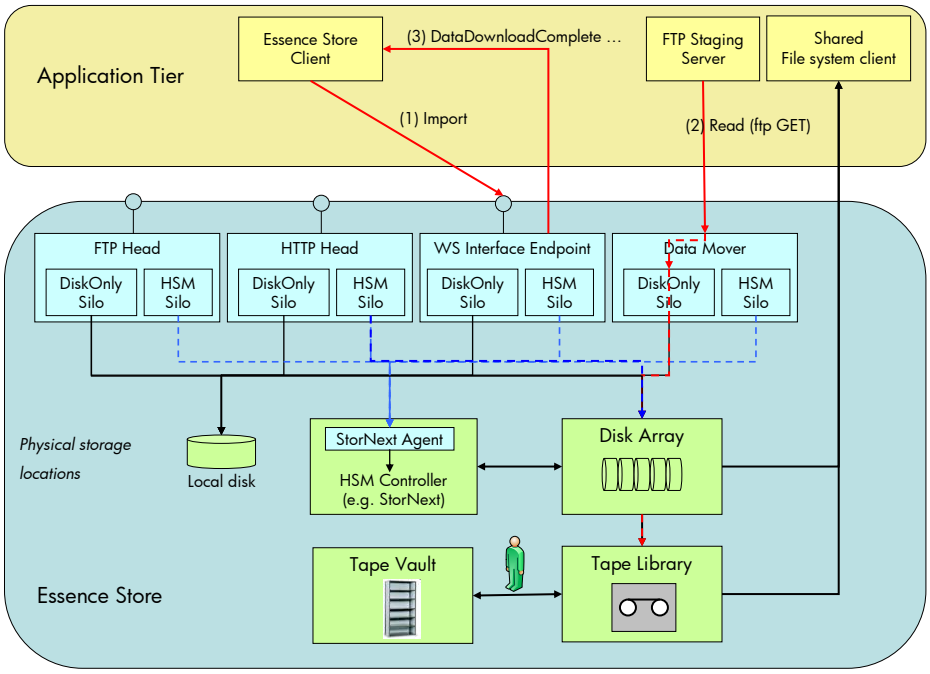

Figure 14: Import Operation

8.3 Essence Store Implementation Technologies

The Essence Store implementation uses the following technologies:

- Linux Red Hat Advanced Server (AS) 4.0 as operating system.
- Java 2 Platform 1.5.0_10 as the programming language.
- J2EE application server: BEA WebLogic Server 9.2 as web service platform.
- Oracle 10g Release 2 as database management system.

- HP StorageWorks EVA SAN storage arrays for online storage.
- HP StorageWorks ESL E-Series Tape Libraries for nearline storage.
- StorNext file system as HSM system.
- Jetty as bulk http access proxy.
- The Linux High Availability (HA) project (heartbeat) [4]
- Object-relational mapping was achieved using Hibernate [7].

8.4 Event Types

There are two types of event fulfilment events and lifecycle events.

Fulfilment Events

These events should be processed by client solutions, so that they can monitor the progress of long running asynchronous jobs (e.g. Import).

DataUploadCompleteEvent

Event sent when the uploading part of an Export or ImportExport is complete. This means that the data has been uploaded into the destination. However, the job may not be complete yet.

DataDownloadCompleteEvent

Event sent during an Import or ImportExport, when all the data has been downloaded to the Essence Store. This does not necessarily mean that the import is complete.

ImportCompleteEvent

Event sent when the Import operation has completed. Note that this is sent either in the case of success or failure; the JobStatus field contains the result status. If successful, then the data will now be in the Essence Store, and, if required, backup tapes will have been made.

ExportCompleteEvent

Event sent when the Export operation has completed. Note that this is sent either in the case of success or failure; the JobStatus field contains the result status.

ImportExportCompleteEvent

Event sent when the ImportExport operation has completed. Note that this is sent either in the case of success or failure; the JobStatus field contains the result status. If successful, then the data will now be in the Essence Store, and, if required, backup tapes will have been made. Also, if successful the export will have been completed.

MakeAccessibleCompleteEvent

Event sent when the MakeAccessible operation has completed. Note that this is sent either in the case of success or failure; the JobStatus field contains the result status.

ImportMakeAccessibleCompleteEvent

Event sent when the operation has completed. Note that this is sent either in the case of success or failure; the JobStatus field contains the result status. If successfull, then the data will now be in the Essence Store, and, if required, backup tapes will have been made. Also, if successfull, the requested lease will now be active.

LeaseExtendedEvent

Sent when the expiry time of a lease gets extended.

LeaseExpireWarningEvent

Sent when a lease is due to expire soon.

LeaseExpiredEvent

Sent when a lease expires, either through the expiry date passing, or through the ExpireEvent call. The lease is now no longer active.

LeaseActiveEvent

Sent when the lease becomes active.

LeaseRevokedEvent

Sent when the lease has been revoked by the Essence Store.

Lifecycle Events

The following events are designed to be consumed by administration solution components that want to keep track of changes to essences and changes to jobs. These events can be optionally ignored by the requester should they wish to because all key information is also contained in the fulfillment events described previously.

EssenceCreatedEvent

This event is sent when an essence is first created.

EssenceChangedEvent

This event is sent when an essence is modified as a result of an Import or ImportMakeAccessible operation.

EssenceDeletedEvent

This event is sent when an essence is deleted from the Essence Store.

JobChangedEvent

This event is sent when a job has changed state. Since most of the long running operations results in several processing stages, there are several JobChangedEvents that are sent out during the execution of the operation.

LeaseCreatedEvent

This event is sent when a Lease is created for an essence as part of a MakeAccessible or ImportMakeAccessible operation call.