The International Journal of Digital Curation
Issue 1, Volume 5 | 2010

An Emergent Micro-Services Approach to Digital Curation Infrastructure

Stephen Abrams, John Kunze, David Loy,
California Digital Library,
University of California

Abstract

In order better to meet the needs of its diverse University of California (UC) constituencies, the California Digital Library UC Curation Center is re-envisioning its approach to digital curation infrastructure by devolving function into a set of granular, independent, but interoperable micro-services. Since each of these services is small and self-contained, they are more easily developed, deployed, maintained, and enhanced; at the same time, complex curation function can emerge from the strategic combination of atomistic services. The emergent approach emphasizes the persistence of content rather than the systems in which that content is managemed, thus the paradigmatic archival culture is not unduly coupled to any particular technological context. This results in a curation environment that is comprehensive in scope, yet flexible with regard to local policies and practices and sustainable despite the inevitability of disruptive change in technology and user expectation.[1]

[1] This article is based on the paper given by the authors at iPRES 2009; received November 2009, published June 2010.
The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. ISSN: 1746-8256 The IJDC is published by UKOLN at the University of Bath and is a publication of the Digital Curation Centre.

SOME RIGHTS RESERVED

Introduction

Information technology and resources have become both integral and indispensable to the pedagogic mission of the University of California (UC). Members of the UC community routinely produce and utilize a wide variety of digital assets in the course of teaching, learning, and research. These assets represent the intellectual capital of the University; they have inherent enduring value and need to be managed carefully to ensure that they will remain available for use by future scholars. Within the UC system the California Digital Library (CDL) UC Curation Center (UC3) has a broad mandate to ensure the long-term usability of the University's digital assets.

UC3 increasingly sees its mission in terms of digital curation, the set of policies and practices focused on maintaining and adding value to a body of trusted digital content for use now and into the indefinite future (Abbott, 2008). Traditionally, preservation and access have been considered disparate activities. Properly, however, they should be seen as complementary functions: preservation focused on ensuring use *over time*, while use depends upon preservation up to a *point in time* (Rusbridge, 2008). Curation is thus an ongoing process of management and enrichment at all stages of the lifecycle of a digital asset (Higgins, 2008). While curation is not solely a technical undertaking – curation success is, for example, highly dependent on important human competencies, analysis, and decision making – a robust infrastructure in which to manage valuable digital content efficiently and effectively is nevertheless a necessary foundation.

Curation Infrastructure

As a central system-wide service provider to the 10 UC campuses, UC3 is continually asked to assume stewardship responsibility for digital content in ever increasing number, size, and diversity of type. Furthermore, this content is often used and repurposed in novel contexts. Thus, the programmatic imperative of UC3 is to provide a curation environment that is comprehensive in scope, yet flexible with regard to local policies and practices, the inevitability of disruptive change in technology and user expectation, and the realization that curation over archival time-spans is a relay (Janée, Frew, & Moore, 2008).

To achieve this goal, UC3 believes it is necessary to deprecate the centrality of the curation repository as *place* (Abrams, Cruse, & Kunze, 2008). The new UC3 approach to digital curation infrastructure is based on the idea of devolving necessary function into a set of independent, but interoperable, micro-services that embody curation values and strategies. Since each of the services is small, they are collectively easier to develop, deploy, maintain, and enhance (Denning, Gunderson, & Hayes-Roth, 2008). Equally as important, since the level of investment in and commitment to any given service is small, they are more easily replaced when they have outlived their usefulness. Although the individual services are narrowly scoped, the complex function needed for effective curation *emerges* from the strategic combination of atomistic services (Fisher, 2006).

Micro-services can be deployed in the contexts in which it makes most sense, both technically and administratively. While UC3 will use the micro-services as the basis for its ongoing centrally-managed curation activities, these services can also be

usefully deployed and operated in local campus IT, research group, and departmental environments. It is no longer necessary that digital content must be transferred to a common repository in order to receive appropriate curation.

Curation Micro-Services

The UC3 curation micro-services are intended to achieve the following strategic goals reflective of evolving community best practice:

- Providing safety through redundancy (embodying the principle that "lots of copies keeps stuff safe"; (Reich & Rosenthal, 2001)).
- Maintaining meaning through description ("Lots of description keeps stuff meaningful").
- Facilitating utility through service ("Lots of services keeps stuff useful").
- Adding value through use ("Lots of uses keeps stuff valuable").

In consequence, the overall infrastructural framework is conceived in terms of an initial set of 12 micro-services arranged in four hierarchical service layers, each building upon the necessary foundational function of lower layers, and approaching curation sufficiency in the aggregate (see Table 1). Although the micro-services are assigned a mode and focus for purposes of classification, in actuality the services have broad applicability throughout the full curation lifecycle (see Figure 1).

Mode	Focus	Layer / micro-service
Curation	*Value*	*Interoperation* • Annotation • Notification
	Service	*Application* • Transformation • Search • Index • Ingest
Preservation	*Context*	*Interpretation* • Characterization • Inventory
	State	*Protection* • Replication • Fixity • Storage • Identity

Table 1. Curation micro-services.

The Protection layer Identity and Storage services are foundational to the entire micro-services framework. The Identity service provides a means by which to persistently and unambiguously distinguish and reference a given unit of curated content. The Storage service provides a secure environment for the persistent management of that content. The Fixity service provides the means to detect damage to the bit-level integrity of managed content, and the Replication service manages the synchronization of content replicas.

Note that the four components of the Protection layer operate on content state without any understanding of what that content represents. The contextual meaning of curated content is managed by the higher-level Interpretation layer. The Inventory service maintains a comprehensive, schema-agnostic metadata catalog for the content managed in the Protection layer. The Characterization service provides an automated means to examine and extract the properties of formatted byte streams underlying managed content that are significant for purposes of curation and preservation analysis, planning, and intervention (Abrams, Morrissey, & Cramer, 2008).

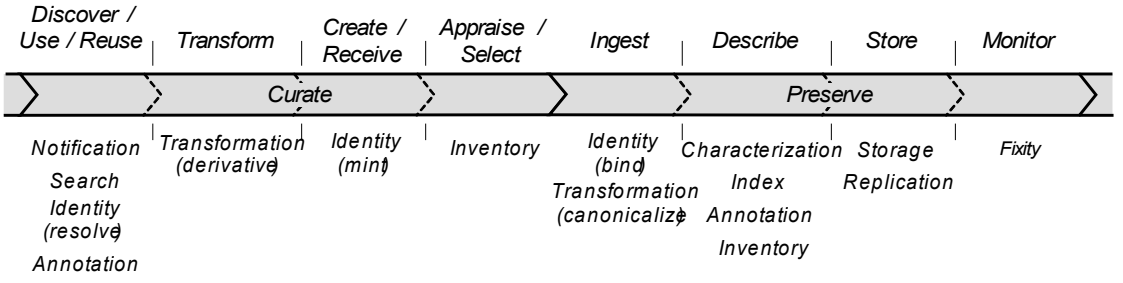

Figure 1. Micro-service lifecycle applicability (adapted from Higgins, 2008).

The Protection and Interpretation layers collectively operate in a back-office preservation mode that would typically be managed directly by repository managers (e.g., UC3 staff). User-facing curation services are provided by the upper two service layers. The Application layer supports base-line functions for both producer and consumer users. The Ingest service provides the means whereby new content is accessioned into the curation environment, with interfaces geared for both manual and automated workflows. The Index and Search services support content and metadata-based search, browse, and retrieval. The Transformation service provides the means to transcode content into desired forms for purposes of ingest canonicalization, preservation migration, and the creation of delivery derivatives.

The upper Interoperation layer supports services for adding value to curated content through consumer-driven use and enrichment. The Notification service provides the means to notify user communities of the availability of newly acquired content. The Annotation service provides the means by which both content curators and consumers can describe the significant properties of content managed in the micro-services infrastructure.

Design Principles

Design of individual curation micro-services is based on the following principles:

- Granularity and orthogonality.
- Complexity through composition rather than addition.
- Persistent interfaces, evolving implementations.
- Flexible configuration, but meaningful deault behavior (the “principle of least surprise”).
- Deferring implementation decision-making until needs and outcomes are clearly understood.

As mentioned previously, complexity is an emergent property of the micro-services approach. In other words, sophisticated curation function arises through the flexible composition of individual, atomistic services rather than through the addition

of function to an increasingly large monolithic system. The continual expansion of the scope of monolithic systems does increase functionality, but at the cost of complexity that complicates development, inhibits maintenance, and increases the likelihood of errant behavior. The UC3 preference is for an aggressive devolution of curation function into simple, focused, independent, but interoperable micro-services.

Micro-services expose their function through well-defined interfaces that define their public service contract (Liegl, 2007; O'Reilly, 2005). Assertions regarding the persistence and sustainability of UC3 curation function are made relative to these interfaces and not their underlying implementations, which can and shall evolve freely over time without invalidating higher-level interface contracts. Interface design is based on the major conceptual entities underlying a given service, which are defined in terms of state properties and behaviors that can access and manipulate that state. Individual state properties are strongly typed and are assigned unique formal identifiers, guaranteed unique within the appropriate scoping unit, so that entity state definitions can be publicly exposed as reusable ontologies.

Abstract interfaces are mapped to three interactive modalities: procedural APIs in various language bindings; command line APIs supported by major operating system command shells; and web APIs conforming to the REST paradigm (Fielding & Taylor, 2002) and incorporating thin client GUIs supported in major browsers (see Figure 2). The intention is to provide content managers and curators with the means to interact with the services without entailing significant changes to established workflows and patterns.

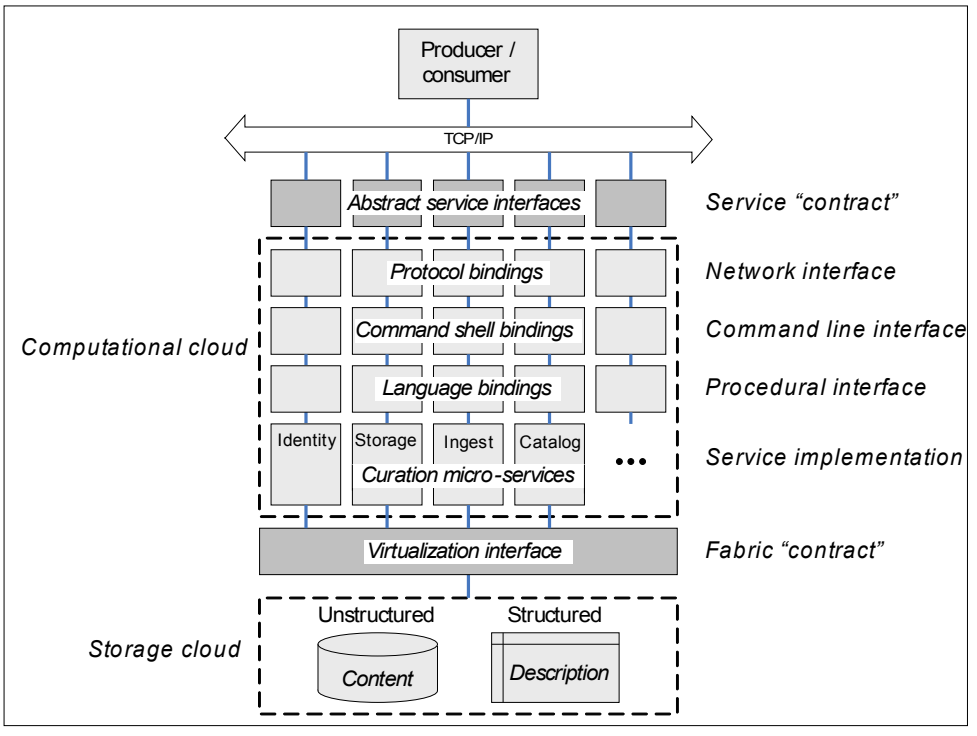

Figure 2. Micro-service stack.

The initial language bindings for the micro-service procedural APIs are Java and Perl. Java RESTful APIs are built with the Jersey framework, the reference implementation of JSR 311, *JAX-RS – Java API for RESTful Web Services*, running in

a Jetty or Tomcat container. The Perl and Java implementations emphasize thin command-line tools that expose as much functionality as feasible to the shell user, but that themselves add minimal functionality to what is already provided by the language-based methods; in this way, maximal function is pushed into the lowest level where it is available in all three modalities.

As an example of these design principles, the Storage service is described in some detail in the following section. As the micro-services are works-in-progress, the apparatus described below does not include some of their more speculative components.

Storage Service

The Storage service manages unstructured storage (i.e., with no common data model) of files holding the digital representations of content. (Structured storage is provided by the Inventory service.) By design the Storage service is opaque with respect to the underlying semantics of stored content, which is managed by the higher-level Inventory service. Consequently, the Storage service has a weak definition of a digital object, which is simply a set of related files descending from a single directory whose state can be modified over time through a sequence of discrete versions. By policy, UC3 strengthens this with the requirement that the directory hierarchy contain every non-derivative file related to the digital object.

Conceptual Modeling.

The *Storage* service is based on five conceptual entities, each defined in terms of its state properties and state manipulating behaviors.

- *Service*. The *Storage* service itself. The *Storage* service acts as a central broker to a number of defined storage nodes, which can be defined for administrative or technical convenience. Global service state includes:
  - Service name, identifier, and version.
  - Enumeration of storage nodes.
  - Number of objects, versions, and files.
  - Total size.
  - Access and support URIs.

  The service encompasses an arbitrary number of storage nodes.

- *Node*. An entity responsible for managing a subset of content known to the service. Nodes are typically defined on the basis of their underlying storage technology or policy regime. Node state includes:
  - Node name, identifier, and version
  - Number of objects, versions, and files.
  - Total size.
  - Storage media: mangetic-disk, magnetic-tape, optical-disk, solid-state.
  - Access modality: on-line, near-line, off-line.
  - Access and support URIs.

  A storage node encompasses an arbitrary number of digital objects.

- *Object*. A set of versioned files representing an intellectually coherent unit of content. Object state includes:
  - Object identifier.
  - Enumeration of versions.
  - Number of versions and files.
  - Total size.
  - Creation, modification, last verification, and last access date/'timestamps.
  - Access URI.

  An object encompasses an arbitrary number of versions.

- *Version*. A set of files representing the discrete state of a digital object at a point in time. Version state includes:
  - Version identifier.
  - Number of files.
  - Total size.
  - Creation, modification, last verification, and last access date/timestamps.
  - Access URI.

  Version identifiers are assigned in numerical sequence, starting with 1. The reserved version number 0 references no fixed version, but is set aside as an access synonym that always represents the current version. A version encompasses an arbitrary number of files.

- *File*. A named digital octet stream. Note that a file octet stream is named, but not typed; the Storage service is not concerned with the *meaning* of the abstract content expressed as a digital object. File state includes:
  - File identifier.
  - Size.
  - Creation, modification, last verification, and last access date/timestamp.
  - Access URI.

Methods.

The Storage service supports a number of methods for accessing and manipulating the conceptual entities and their state. Each method is classified according to the important transactional properties of idempotency and safety (Fielding et al., 1999).

- *Help* [idempotent, safe]
- *Get-service-state* [idempotent, safe]
- *Get-node-state* [idempotent, safe]
- *Get-object* [idempotent, unsafe]
- *Get-object-state* [idempotent, safe]
- *Get-version* [idempotent, unsafe]
- *Get-version-state* [idempotent, safe]
- *Get-file* [idempotent, unsafe]
- *Get-file-state* [idempotent, safe]
- *Add-version* [non-idempotent, unsafe]
- *Delete-object* [idempotent, unsafe]
- *Delete-version* [idempotent, unsafe]

The *Help* method is common to all micro-services and provides a brief descriptive text, an enumeration of all supported methods, and a support contact URI. The *Get-object*, *Get-version*, and *Get-file* methods are trivially unsafe since they modify their respective states with a current access timestamp. Note that the only mechanism for modifying an object's content is to introduce a new version. The *Delete-object* and *Delete-version* methods are defined for completeness, but as a matter of policy are intended for use only in response to unusual curatorial circumstances.

Each method is first defined abstractly and then mapped to specific protocols. For example, the *Get-file-state* method definition is summarized in Table 2. This abstract method definition is mapped to the concrete syntax specified by the web, command line, and procedural APIs, as shown, for example in Figure 3. All implementation details are hidden behind the interface, which constitutes the public service contract. The supported response forms for which state information can be requested are ANVL (Kunze, Kahle, Masanes, & Mohr, 2005), HTML, JSON, RDF/Turtle, and XML.

Parameter	Type	Obligation	Description
Node	Identifier	Mandatory	Storage node
Object	Identifier	Mandatory	Object identifier
Version	Identifier	Mandatory	Version identifier
File	Identifier	Mandatory	File identifier
Form	Enum	Optional	Response form
RETURN	State	Mandatory	File state
SIDE EFFECTS	*Not applicable*		
ERRORS	Badly formed request		
	Node not found		
	Object not found		
	Version not found		
	File not found		
	Unsupported response form		

Table 2. *Get-file-state* method.

```
Get /fileState/node/object/version/file HTTP/1.1
Accept: application/json

% store getFileState node object version file -f json

File.getState(node, object, version, file, Form.JSON);
```

Figure 3. *Get-file-state* method syntax.

Implementation.

The general micro-services principles of granularity and orthogonality are applied throughout the implementation process. Consequently the *Storage* service relies on a number of subsidiary specifications, conventions, and systems (described in more detail at <http://www.cdlib.org/inside/diglib/>).

The Storage service is instantiated in a file system as shown in Figure 4. The file of the form "0=*name_version*" is a Namaste tag (Name-as-text), that functions as an indicative signature of the Storage service home directory; in this case it specifies that this instantiation conforms to version 0.8 of the service specification. The "admin/" directory holds various administrative declarations and the "log/" directory holds access and diagnostic logs. The global state properties of the service are defined by the file "store-info.txt" (see Figure 5).

The storage nodes known to the service are defined by name and access URI in the file "nodes.txt" (see Figure 6). Nodes can be remote or local to the host running the Storage service. Local interoperability assumes that the storage node is instantiated in a file system mountable by the local host; remote nodes are accessed over a TCP/IP network through their access URIs.

```
store_home/
           0=store_0.7
           admin/
           log/
           nodes.txt
           store-info.txt
```

Figure 4. Storage service file system structure.

```
Name: store
Service-scheme: Store/0.7
Node-scheme: CAN/0.8
Verify-on-read: true
Verify-on-write: true
Access-uri: http://store.cdlib.org/
Support-uri: email:store-support@cdlib.org
```

Figure 5. Storage file properties file.

```
can01 http://can01.cdlib.org/
can02 http://can02.cdlib.org/
can03 file:///home/can03
```

Figure 6. Storage nodes file.

The default implementation for a storage node is a Content Access Node (CAN, see Figures 7 and 8), which is essentially a repository instance. A CAN manages its objects in a hierarchical file system tree. The primary convention for the structure of the branches of the tree is Pairtree, which uses a bigram decomposition of an object's identifier to determine the directory hierarchy at which the object's content is found. Thus, an object with identifier "abc123" would be found at the end of the relative directory path "ab/c1/23". (Pairtree defines escaping rules to prevent collision between identifier characters and file system semantics.) Consistent with the principle of micro-service independence, Pairtree has been adopted by a number of external institutions and initiatives. For example, it is being used by HathiTrust (York, 2009) to store millions of scanned books. Open source Perl code supporting Pairtree, Namaste, and ANVL are available (Kunze, 2009).

```
can_home/
         0=can_0.8
         admin/
         can-info.txt
         log/
         store/
               pairtree_root/
                              0=pairtree_0.1
                              pairtree-info.txt
                              ab/
                                 c1/
                                    23/
                                       abcd123/
```

Figure 7. CAN file system structure.

The leaf at the end of a Pairtree path stores the digital object, but its nature is not specified by Pairtree. For example, it could be a Bagit bag (Boyko, Kunze, Littman, Madden, & Vargas, 2009), a HathiTrust digitized book, or a web crawl. In the context of a CAN, the convention controlling the structure of that object is Dflat.

```
Name: can01
Node-scheme: CAN/0.8
Branch-scheme: Pairtree/0.1
Leaf-scheme: Dflat/0.16
Media-type: magnetic-disk
Access-mode: on-line
Verify-on-read: true
Verify-on-write: true
Access-uri: http://can01.cdlib.org/
```

Figure 8. CAN properties file.

Dflat defines structures for managing versioned sets of files that represent a digital object (see Figures 9 and 10). Object versions are stored in numbered directories of the form "v*nnn*/". (Directory names corresponding to versions numbered up to 999 are left-padded to align lexical and numeric ordering; names above 999 naturally extend an additional digit per order of magnitude.) The symbolic link "current@" provides direct access to the current version.

```
dflat_home/
            0=dflat_0.16
            admin/
            current@
            dflat-info.txt
            log/
            v001/
            v002/
            V003/
```

Figure 9. Dflat file system structure.

```
Object-scheme: Dflat/0.16
Manifest-scheme: Checkm/0.1
Full-scheme: Dnatural/0.12
Delta-scheme: ReDD/0.1
Current-scheme: symlink
```

Figure 10. Dflat properties file.

A CAN is a container for everything that might belong in a repository instance. While its specification is still evolving, it bundles the premises that a CAN repository collection is represented by one or more Pairtrees and that the leaves of each Pairtree are Dflats. Consistent with stated design principles, some implementation decision-making has been deferred until needs are more clearly understood; currently absent are ways to represent policies governing such things as frequency of fixity checking, remote replication sites, admissibility of annotations, etc.

A Dflat version can be represented in fully-instantiated or delta-compressed form. The current version is always fully instantiated; all previous versions are generally kept in delta-compressed form to minimize storage utilization. Regardless of representation type, all version directories hold a manifest file ("manifest.txt") conforming to the Checkm specification, which associates a size and message digest with each version file.

The structure of a fully-instantiated version representation is defined by the subsidiary Dnatural convention (see Figure 11). The content data and metadata received from an object's producer or curator are stored in the "data/" and "metadata/" directories, respectively. The content of the "data/" directory is completely up to the producer or curator (e.g., it could be a BagIt bag). The "enrichment/" directory holds additional metadata and derivative content automatically generated by the Storage service itself. The "annotation/" directory holds additional metadata and derivative content supplied by content consumers.

```
v003/
     0=dnatural_0.12
     admin/
     annotation/
     data/
     enrichment/
     manifest.txt
     metadata/
```

Figure 11. Dnatural file system structure.

Compressed version representations conform to the Reverse Directory Delta (ReDD) convention (see Figure 12). ReDD is a very simple tool- and platform-independent scheme that uses file-level reverse deltas to minimize overall storage utilization. The "add/" directory holds the files that need to be added relative to the subsequent version in order to re-instantiate the previous version; the "delete.txt" file lists the files that need to be deleted relative to the subsequent version to re-instantiate the previous version. In other words, the delta information associated with version 2 indicates how to manipulate the files of version 3 in order to recover the complete form of version 2. Access is thus faster for later versions; the re-instantiation of a version early in the chronological sequence will require the iterative application of deltas. Dflat maintains an ordered sequence of versions, and can be applied to any differencing scheme (e.g., Unix "diff") that operates on the notions of current and previous version.

```
v002/
     0=redd_0.1
     d-manifest.txt
     delta/
           add/
           delete.txt
     manifest.txt
```

Figure 12. ReDD file system structure.

All of the conventions and subsystems underlying the Storage service are supported by procedural APIs in separate package spaces, so they can easily be repurposed in other contexts. The reliance on the file system as the paradigmatic storage abstraction is justified by the design and behavioral characteristics of modern file systems such as ZFS (Bonwick & Moore, 2007), which exhibits essentially constant time read and write performance independent of total number or size of files (Abrams, Cruse, Kunze, & Loy, 2009). The Storage service as deployed by UC3 policy will serve as the copy of record for all information known about a unit of digital content. While a subset of this information will be managed in structured form by the Inventory service, this is considered to be a duplicative, rather than authoritative copy, for purposes of optimizing routine administrative and curatorial queries. If necessary, the Inventory service can be fully re-instantiated through an exhaustive traversal of various Storage service file system structures.

All of the micro-service implementations are designed to be fully self-contained and easily deployed and operated with minimal human intervention. While UC3 will continue to provide a centrally-managed curation repository, the intention of the micro-services approach is to facilitate the distribution of efficient and effective

curation function to new constituencies and contexts, including campus data centers, academic departments, laboratory and field station computing clusters, and scholars' desktops.

Development Process

Establishing the UC3 micro-services infrastructure draws from both traditional and agile development principles:

- *An engaged user community driving needs assessment and functional requirements*
- *Early prototyping with frequent refactoring.*
- *Continuous build and test.*
- *Documentation as a co-deliverable, not an afterthought.*
- *A small group of early adopters.*

The 12 micro-services are being implemented in a sequence of six developmental waves (see Table 3). The second, fourth, and sixth wave represent significant deliverable milestones, corresponding to a minimally, moderately, and fully functional curation repository, respectively.

First wave	Second wave ✓	Third wave	Fourth wave ✓	Fifth wave	Sixth wave ✓
Identity	*Inventory*	*Index*	*Search*	*Notification*	*Annotation*
Storage	*Ingest*	*Fixity*	*Replication*	*Characterization*	*Transformation*
Object / collection modeling			*Authentication / authorization*		
Policy and business model development					

Table 3. Micro-services developmental waves.

The first through third waves are accompanied by the concomitant development of standards and conventions for modeling digital objects and object collections. The fourth through sixth waves will be accompanied by the development of common authentication and authorization mechanisms. All six waves will be accompanied by policy and business modeling.

The Identity and Storage services are currently available in working form; the second wave milestone deliverables will be ready to accept content in January 2010. The initial content will be provided by a multi-campus pilot project on the curation of electronic theses and dissertations. Subsequent content projects will involve anthropological and zoological museum collections, environmental field data, and biological type specimens.

Conclusion

In order to facilitate the application of UC Curation Center service offerings to new campus constituencies, and the increasing number, size, and type diversity of digital content, the underlying curation infrastructure must be easily adaptable to local needs and practices. An architectural approach in which curation function is embodied in a set of granular and orthogonal micro-services best provides the necessary deployment flexibility, while also simplifying development and maintenance effort. Service interoperability is facilitated by strict conformance to the behavioral semantics of well-defined public interfaces. This permits comprehensive curation function to emerge from the strategic combination of individual atomistic services.

References

Abbott, D. (2008). *What is digital curation?* Retrieved September 7, 2009, from http://www.dcc.ac.uk/resource/briefing-papers/what-is-digital-curation/

Abrams, S., Cruse, P., & Kunze, J. (2008). Preservation is not a place. *International Journal of Digital Curation, 4*(1), 8-21. Retrieved September 7, 2009, from http://www.ijdc.net/index.php/ijdc/article/view/98/73

Abrams, S., Cruse, P., Kunze, J., & Loy, D. (2009). "Where are we from? Where are we going?": Permanent objects, disposable systems. *4th International Conference on Open Repositories*, Atlanta, May 27-29, 2009.

Abrams, S., Morrissey, S., & Cramer, T. (2008). "What? So what?": The next-generation JHOVE2 architecture for format-aware characterization. *Fifth International Conference on Preservation of Digital Objects*, London, September 29-30, 2008.

Bonwick, J., & Moore, B. (2007). *ZFS: The last word in file systems*. Retrieved September 7, 2009, from http://www.opensolaris.org/community/zfs

Boyko, A., Kunze, J., Littman, J., Madden, L., & Vargas, B. (2009). *The BagIt File Packaging Format.* Retrieved September 7, 2009, from http://www.cdlib.org/inside/diglib/bagit/bagitspec.html

Denning, P. J., Gunderson, C., & Hayes-Roth, R. (2008). Evolutionary system development. *Communications of the ACM, 51*(17), 29-31.

Fielding, R., Gettys, J., Mogul, J., Frystuk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616. Retrieved September 7, 2009, from http://www.ietf.org/rfc/rfc2616.txt

Fielding, R., & Taylor, R. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology, 2*(2), 115-150.

Fisher, D. A. (2006). *An emergent perspective on interoperation in systems of systems*. Technical Report, CMU/SEI-2006-TR-003, ESC-TR-2006-003, Carnegie-Mellon University. Retrieved September 7, 2009, from http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr003.pdf

Higgins, S. (2008). The DCC curation lifecycle model. *International Journal of Digital Curation, 3*(1), 134-140. Retrieved September 7, 2009, from http://www.ijdc.net/index.php/ijdc/article/view/69/48

Janée, G., Frew, J., & Moore, T. (2008). Relay-supporting archives: Requirements and progress. *International Journal of Digital Curation 4*(1*)*, 57-70. Retrieved September 7, 2009, from http://www.ijdc.net/index.php/ijdc/article/view/102/77

Kunze, J. (2009). *Software modules, command-line scripts, test suites, and documentation supporting Pairtree, Namaste tags, and ANVL*. Comprehensive Perl Archive Network. Retrieved September 7, 2009, from http://search.cpan.org/~jak

Kunze, J., Kahle, B., Masanes, J., & Mohr, G. (2005). *A Name-Value Language (ANVL)*. Retrieved September 7, 2009, from http://www.cdlib.org/inside/diglib/ark/anvlspec.pdf

Liegl, P. (2007). The strategic impact of service oriented architectures. *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, Arizona, March 26-29, 2007.

O'Reilly, T. (2005). *Web 2.0: Design patterns and business models for the next generation of software*. Retrieved September 7, 2009, from http://oreilly.com/web2/archive/what-is-web-20.html

Reich, V., & Rosenthal, D. S. H. (2001, June). LOCKSS: A permanent web publishing and access system. *D-Lib Magazine, 7*(6). Retrieved September 7, 2009, from http://www.dlib.org/dlib/june01/reich/06reich.html

Rusbridge, C. (2008, July 29). Re: "Digital preservation" term considered harmful? [Web log message from *Digital Curation Blog*]. Retrieved September 7, 2009, from http://digitalcuration.blogspot.com/2008/07/digital-preservation-term-considered.html

York, J. (2009). This library never forgets: Preservation, cooperation, and the making of HathiTrust digital library. In *Archiving 2009 Final Program and Proceedings.* Retrieved September 7, 2009, from http://www.hathitrust.org/documents/This-Library-Never-Forgets.pdf