Preservation of Digital Data with Self-Validating, Self-Instantiating Knowledge-Based Archives*

Bertram Ludäscher Richard Marciano Reagan Moore

San Diego Supercomputer Center, U.C. San Diego

{ludaesch,marciano,moore}@sdsc.edu

Abstract

Digital archives are dedicated to the long-term preservation of electronic information and have the mandate to enable sustained access despite rapid technology changes. Persistent archives are confronted with heterogeneous data formats, helper applications, and platforms being used over the lifetime of the archive. This is not unlike the interoperability challenges, for which mediators are devised. To prevent technological obsolescence over time and across platforms, a migration approach for persistent archives is proposed based on an XML infrastructure.

We extend current archival approaches that build upon standardized data formats and simple metadata mechanisms for collection management, by involving high-level conceptual models *and* knowledge representations *as an integral part of the archive and the ingestion/migration processes. Infrastructure independence is maximized by archiving generic,* executable specifications *of (i) archival constraints (i.e., "model validators"), and (ii) archival transformations that are part of the ingestion process. The proposed architecture facilitates construction of* self-validating *and* self-instantiating *knowledge-based archives. We illustrate our overall approach and report on first experiences using a sample collection from a collaboration with the National Archives and Records Administration (NARA).*

1 Background and Overview

Digital archives have the mandate to capture and preserve information in such a way that the information can be (re-)discovered, accessed, and presented at any time in the future. An obvious challenge for archives of digital information is the limited storage lifetimes due to physical media decay. However, since hardware and software technologies evolve rapidly and much faster than the media decay, the real challenge lies in the *technological obsolescence* of the infrastructure that is used to access and present the archived information. Among other findings, the Task Force on Archiving Digital Information concluded that an infrastructure is needed that supports distributed systems of digital archives, and identified *data migration* as a crucial means for the sustained access to digital information [5].

In a collaboration with the National Archives and Records Administration (NARA), the San Diego Supercomputer Center (SDSC) developed an information management architecture and prototype for digital archives, based on scalable archival storage systems (HPSS), data handling middleware (SRB/MCAT), and XML-based mediation techniques (MIX) [9, 12, 1].[1]

To achieve the goal of reinstantiating archived information on a future platform, it is not sufficient to merely copy data at the bit level from obsolete to current media but to create "recoverable" archival representations that are *infrastructure independent* (or *generic*) to the largest extent possible. Indeed the challenge is the forward-migration in time of *information and knowledge about* the archived data, *i.e.*, of the various kinds of meta-information that will allow recreation and interpretation of structure and content of archived data.

In this paper, we describe an architecture for infrastructure independent, *knowledge-based* archival and collection management. Our approach is knowledge-based in the sense that the ingestion process can employ both structural and semantic models of the collection, including a "flattened" relational representation, a "reassembled" semistructured representation, and higher-level "semantic" representation. The architecture is *modular* since the ingestion process consists of transformations that are put together and executed

*This research has been sponsored by a NARA supplement to the National Science Foundation project ASC 96-19020.

[1] www.clearlake.ibm.com/hpss/, www.npaci.edu/DICE/SRB, and www.npaci.edu/DICE/MIX/

in a *pipelined* fashion. Another novel feature of our approach is that we allow *archiving of the entire ingestion pipeline*, *i.e.*, the different representations of the collection together with the transformation rules that were used to create those representations.

The organization is as follows: In Section 2 we introduce the basic approaches for managing technology evolution, in particular via migratable formats. Section 3 presents the elements of a fully XML-based archival infrastructure. In Section 4, we show how this architecture can be extended to further include conceptual-level information and knowledge about the archived information. A unified perspective on XML-based "semantic extensions" is provided by viewing them as *constraint languages*. The notions of *self-validating* and *self-instantiating* archives are given precise meanings based on a formalization of the ingestion process. We report on first experiences using a real-world collection in Section 5 and conclude in Section 6.

2 Managing Technology Evolution via Migratable Formats

As long as digital objects "live" in their original (non-archival) runtime environment, they are permanently threatened by technological obsolescence. Here, by *digital object* we mean a machine readable representation of some data, an image of reality, or otherwise relevant piece of information in some recognizable data format (*e.g.*, an electronic record of a Senator's legislative activities in the form of an RTF[2] text file, a row of data in a scientific data set[3], or a MIME-encoded email containing text and MPEG-7 encoded multimedia objects). The data format in which the digital object is encoded can contain *metadata* that either explicitly or implicitly (*e.g.*, via reference to standards) describe further information about the data such as structure, semantics, context, provenance, and display properties. Metadata can be embedded via inlined markup tags and attributes or may be packaged separately. Since metadata is also data, *its* meaning, context, *etc.* could be described via meta-metadata. In practice, however, this loop is terminated by assuming the use of agreed-upon metadata standards. In Section 4.2 a more self-contained approach is presented that includes *executable specifications* of semantic constraints as part of archival packages.

[2]Microsoft's Rich Text Format

[3]`www.faqs.org/faqs/sci-data-formats/`

Digital objects are handled by *helper applications* (*e.g.*, word processors, simulation tools, databases, multimedia players, records management software, *etc.*) that interpret the objects' metadata to determine the appropriate processing steps and display actions. Helper applications run on an underlying *operating system* (OS) that ultimately executes presentation and interaction instructions. As time goes by, new versions of data formats, their associated helper applications, or the underlying OS increase the risk of technological obsolescence and can cause the digital information to perish. There are several approaches to avoid such loss of accessibility to stored information:

(1) If a new version of the OS is not backward compatible, *patch* it in such a way that the old helper applications still runs. This is impossible for proprietary operating systems, but (at least in principle) feasible for open, non-proprietary ones such as Linux.

(2) Put a wrapper around the new OS, effectively *emulating* (parts of) the old OS such that the unchanged helper application can still run.

(3) *Migrate the helper application* to a new OS; also ensure *backward compatibility* of new versions of the helper application.

(4) *Migrate the digital objects to a new format* that is understood by the new versions of helper applications and OS.

One can refine these basic approaches, *e.g.*, by decoupling helper applications from the OS via an intermediary *virtual machine* VM. Then helper applications can run on different platforms simply by migrating the VM to those platforms. For example, the Java VM is available for all major platforms, so helper applications implemented on this VM can be run anywhere (and anytime) where this VM is implemented.[4]

Note that (4) assumes that the target format is understood by the new helper application. By requiring the migration target in (4) to be a *standard format* (such as XML), this will indeed often be the case. Moreover, standards change at a lower rate than arbitrary formats, so *conversion frequency* and thus migration cost is minimized.

The different ways to manage technology evolution can be evaluated by comparing the tension

[4]VMs are not new – just check your archives! Examples include the P-Code machine of UCSD Pascal and the Warren Abstract Machine (WAM), the target of Prolog compilers.

induced by retaining the original presentation technology and the cost of performing conversion (Table 1): information discovery and presentation capabilities are much more sophisticated in newer technologies. When access and presentation of information is limited to old technologies, the more efficient manipulation provided by new tools cannot be used.[5]

	Technology Tension	*Conversion Frequency*
(1)	old display format	OS update
(2)	old display format	OS update
(3)	old display format	helper app. update
(4)	new display tools	new standard

Table 1. Tracking technology evolution

In the sequel, we focus on (4), a *migration through self-describing standards* approach which aims at minimizing the dependency on specific hardware, software (OS, helper apps.), data formats, and references to external, non-persistently archived resources (relevant contextual or meta-information that cannot be inlined should be indirectly included via references to persistently archived information). Hence infrastructure independence is maximized by relying on interoperability technologies. It is not coincidental that XML not only provides a "good archival format" (minimizing migration cost), but also a good data exchange and interoperability framework: a persistent archive has to support evolving platforms and formats over time. This is similar to supporting different platforms at one time. In other words, a persistent archive can be seen as an interoperability system.

3 XML-Based Digital Archives

In this section, we describe the basic processes and functions of digital archives (using concepts and terminology from the Open Archival Information System (OAIS) reference model [11]), and show how an infrastructure for digital archives can be built around XML standards and technologies.

Functional Architecture. The primary goal of digital archives is long-term preservation of information in a form that guarantees sustained access and dissemination in the future (Fig. 1): Initially, the information producer and the archive need

[5]Restricting the presentation capabilities to the original technology is analogous to reading a 16^{th} century book only in flickering candlelight.

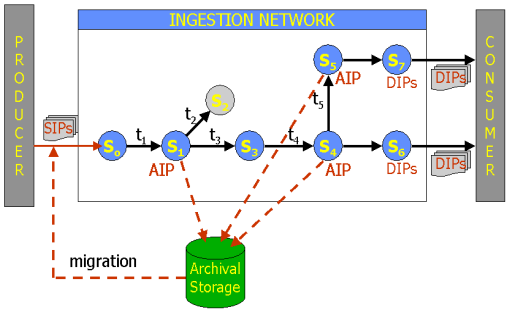

Figure 1. Digital archive architecture

to agree on the *submission* or *accessioning policies* (*e.g.*, acceptable submission formats, specifications on what is to be preserved, access functions, and other legal requirements). Subsequently, the producer can transfer *submission information packages* (SIPs) to the archive, where they enter – in our proposed system – an *ingestion network* (see Definition 3). An initial *quality assurance* check is performed on SIPs and corresponding feedback returned to the producer. As the SIPs are transformed within the ingestion network, *archival information packages* (AIPs) are produced and put into *archival storage*.

Migration of AIPs is a normal "refreshing" operation of archives to prevent the obsolescence of AIPs in the presence of changes in technology or the contextual information that is necessary to interpret the AIPs. Indeed, failure to migrate AIPs in a timely manner jeopardizes the creation of *dissemination information packages* (DIPs) at a later time. Migration may be seen as a feedback of the AIPs into an *updated* ingestion network that produces AIPs for new technology from the previous AIP formats (in this sense, the initial ingestion can be viewed as the 0^{th} migration round). Creation of DIPs from AIPs is sometimes called *(re)-instantiation* of the archived collection. Observe that "good" AIP formats aim to be maximally self-contained, self-describing, and easy to process in order to minimize migration and dissemination cost.

The ingestion and migration *transformations* can be of different nature and involve data *reformatting* (*e.g.*, physical: from 6250 to 3480 tape, or bit-level: from EBCDIC to ASCII), and data *conversion* (*e.g.*".rtf" to ".html"). Conversions have to be content-preserving and should also preserve as much structure as possible. However, some-

times content is "buried" in the structure and can be lost accidentally during an apparently "content-preserving" conversion (Section 5). For dissemination and high-level "intelligent access", further transformations can be applied, *e.g.*, to derive a *topic map* view [14] that allows the user to navigate data based on high-level concepts and their relationships. Documentation about the sequence of transformations that have been applied to an AIP is added to the *provenance* metadata.

Digital archives implementing the above functions and processes can be built upon the following system components: (i) *archival storage systems* (*e.g.*, HPSS), (ii) *data management systems* (*e.g.*, XML-generating wrappers for data *ingestion*; databases and XML processors for querying and transforming collections during *migration*), and (iii) *data presentation systems* (*e.g.*, based on web services with style sheets, search interfaces and navigational access for *dissemination*).

OAIS Information Packages. In order to understand the disseminated information, the consumer (Fig. 1) needs some initial, internal knowledge (*e.g.*, of the English language). Beforehand, the presentation system must be able to interpret the AIP, i.e., it needs *representation information* which is packed together with the actual content. Intuitively, the more representation information is added to a package, the more self-contained it becomes. According to the OAIS framework [11], an *information package* IP contains *packaging information* PI (*e.g.*, the ISO-9660 directory information of a CD) that encapsulates the actual *content information* CI and additional *preservation description information* PDI. The latter holds information about the associated CI's *provenance* PR (origin and processing history), *context* CON (relation to information external to the IP), *reference* REF (for identifying the CI, say via ISBN or URI), and *fixity* information FIX (*e.g.*, a checksum over CI). Finally, similar to a real tag on a physical object, the IP has *descriptive information* DI on the "outside" that is used to discover which IP has the CI of interest. Put together, the encapsulation structure of an IP is as follows [11]:

- IP = [DI [PI [CI PDI[PR CON REF FIX]]]] (*)

Information Hierarchy. Information contained in an archive or IP can be classified as follows: Above the *bitstream*, *character*, and *word levels*, we can identify individual digital objects as tuples, records, or similar object structures. We call this the *data-* or *instance-level*. Such object-level information is packaged into the CI. At the *schema-* or *class-level*, structural and type information is handled: this metadata describes types of object attributes, aggregation information (collections/subcollections), and further descriptive *collection-level metadata* (*e.g.*, SDSC's Storage Resource Broker (SRB/MCAT)[6] provides a state-of-the-art "collection-aware" archival infrastructure). Collection-level metadata can be put into the PI and the DI. Finally, information at the *conceptual-level* captures *knowledge* about the archive and includes, *e.g.*, associations between concepts and object classes, relationships between concepts, and derived knowledge (expressed via logic rules). While some of this knowledge fits into the CON package, we suggest to provide a distinct *knowledge package* KP as part of the PDI. Possible representations for expressing such knowledge range from database-related formalisms like (E)ER diagrams, XML Schema, and UML class diagrams, to AI/KR-related ones like logic programs, semantic networks, formal ontologies, description logics, to recent web-centric variants like RDF(-Schema), DAML(+OIL) taken in tow by the Semantic Web boat [2]. Some of these formalisms have been standardized already or will become in the near future, hence are candidate archival formats (*e.g.*, XMI which includes UML model exchange and OMG's Meta Object Facility MOF [15], RDF [13], the Conceptual Graph Standard [3] and the Knowledge-Interchange Format [6]). While the number of possible formalisms (and the complexity of some of them) makes this a daunting task for any archival system, there is hope: most of them are based on predicate logic or "classic" extensions (*e.g.*, rule-based extensions of first-order predicate logic have been extensively studied by the logic programming and deductive databases communities). Based on a less volatile "standard" logic framework, one can devise generic, universal formalisms that can express the features of other formalisms in a more robust way using *executable specifications* (see below).

XML-Based Archival Infrastructure. It is desirable that archival formats do not require special access software and be standardized, open, and as simple as possible. Ideally, they should be self-contained, self-describing "time-capsules". The specifications of proprietary formats[7] like Word,

[6] `www.npaci.edu/DICE/SRB`

[7] Proprietary formats like ".doc" tend to be complex, undocumented, and "married" to a hardware or software envi-

Wordperfect, *etc.* may vary from version to version, may not be available at all, or — even if they are available (*e.g.*, RTF) — may still require specialized helper applications ("viewers") to interpret and display the information contained in a document. Similary, formats that use data compression like PDF require that the specification describes the compression method and that this method is executable in the future. XML, on the other hand, satisfies many desiderata of archival: the language is standardized [16], and easy to understand (by humans) and parse (by programs). Document structure and semantics can be encoded via user-definable tags (*markup*), sometimes called *semantic tags*: unlike HTML, which contains a fixed set of structural and presentational tags, XML is a language for defining new languages, *i.e.*, a metalanguage. Because of user-defined tags, and the fact that documents explicitly contain some schema information in the structure of their parse tree (even if a DTD or XML Schema is not given), XML can be seen as a *generic, self-describing* data format.

Viewed as a data model, XML corresponds to labeled, ordered trees, *i.e.*, a *semistructured data model*. Consequently, XML can easily express the whole range from highly structured information (records, database tables, object structures) to very loosely structured information (HTML, free text with some markup). In particular, the structure of an information package IP as indicated in (*) above can be directly represented with *XML elements*: IPs (and contained sub-IPs) are *encapsulated* via delimiting opening and closing tags; descriptive (meta)-information DI about a package can be attached in XML *attributes*, *etc.*XML elements can be *nested* and – since order of subelements is preserved – ordered and unordered collection types (*list*, *bag*, *set*) can be easily encoded, thereby directly supporting collection-based archives.

The core of our archival architecture is the ingestion network. Some distinguished nodes (or *stages*) of the ingestion net produce AIPs, others yield different "external views" (DIPs). As IPs pass from one stage to the next, they are queried and restructured like database instances. At the syntactic level, one can maximize infrastructure independence by representing the databases in XML and employing standard tools for parsing, querying, transforming, and presenting XML.[8] To ensure *modularity* of the architecture, complex XML transformations should be broken up into smaller ones that can be expressed directly with the available tools. For supporting *huge data volumes* and *continuous streams* of IPs, the architecture needs to be *scalable*. This can be achieved with a *pipelining execution model* using *stream-based* XML transformation languages (*i.e.*, whose memory requirements do *not* depend on the size of the XML "sent over the wire"). As the XML IPs are being transformed in the ingestion net, provenance information PR is added. This includes the usual identification of the organizational unit and individuals who performed the migration, as well as identification of the *sequence of XML mappings* that was applied to the IP. By storing *executable specifications* of these mappings, *self-instantiating* archives can be built (Section 4.3).

4 Knowledge-Based Archives

In this section, we propose to extend the purely structural approach of plain XML to include more *semantic information*. By employing "executable" knowledge representation formalisms, one can not only capture more semantics of the archived collection, but this additional information can also be used to *automatically validate* archives at a higher, conceptual level than before where it was limited to low-level fixity information or simple structural checks.

Intuitively, we speak of a *knowledge-based* (or *model-based*) archival approach, if IPs can contain conceptual-level information in *knowledge packages* (KPs). The most important reason to include KPs is that they capture meta-information that may otherwise be lost: For example, at ingestion time it may be known that digital objects of one class inherit certain properties from a superclass, or that functional or other dependencies exists between attributes, *etc.*Unfortunately, more often than not, such valuable information is not archived explicitly.

During the overall archival process, KPs also provide additional opportunities and means for quality assurance: At ingestion time, KPs can be used to check that SIPs indeed conform to the given accessioning policies and corresponding feedback can be given to the producer. During archival management, *i.e.*, at migration or dissemination time, KPs can be used to *verify* that

ronment. Data formats whose specifications can be "grasped easily" (both physically and intellectually) and for which tools-support is available, are good candidate archival formats.

[8] *e.g.*, SAX, XPath, XQuery, XSLT, ...

the CI satisfies the pre-specified *integrity constraints* implied by the KPs. Such value-added functions are traditionally not considered part of an archival organization's responsibilities. On the other hand, the detection of "higher-level inconsistencies" clearly yields valuable meta-information for the producers and consumers of the archived information and could become an integral service of future archives.

The current approach for "fixing the meaning" of a data exchange/archival format is to provide an XML DTD. For example, many organizations and groups defined their "community language" in this way. However, the fact that a document has been *validated* say wrt. the Encoded Archival Description DTD [4] does *not* imply that it satisfies all constraints that are part of the EAD specification. Indeed, only *structural constraints* can be automatically checked using a (DTD-based) validating parser – all other constraints are not checked at all or require specialized software.

These and other shortcomings of DTDs for data modeling and validation have been widely recognized and have led to a flood of extensions, ranging from the heavyweight, W3C-supported XML Schema proposal [17],[9] to more grassroots efforts like RELAX (which may become a standard) [10],[10] and many others (RDF, RDF-Schema, SOX, DSD, Schematron, XML-Data, DCD, XSchema/DDML, ...). A unifying perspective on these languages can be achieved by viewing them as *constraint languages* that distinguish "good documents" (those that are *valid* wrt. the constraints) from "bad" (*invalid*) ones.

4.1 XML Extensions as Constraint Languages

Assume IPs are expressed in some *archival language* $\mathcal{A}$. In the sequel, let $\mathcal{A} \supseteq$ XML. A concrete *archive instance* (short: *archive*) is a "word" a of the archival language $\mathcal{A}$, *e.g.*, an XML document.

Definition 1 (Archival Constraint Languages) We say that $\mathcal{C}$ is a *constraint language* for $\mathcal{A}$, if for all $\varphi \in \mathcal{C}$ the set $\mathcal{V}_\varphi = \{a \in \mathcal{A} \mid a \models \varphi\}$ of *valid archives* (wrt. φ) is decidable. □

For example, if $\mathcal{C} =$ DTD, a *constraint* φ is a concrete DTD: for any document $a \in$ XML, validity of a wrt. the DTD φ is decidable (so-called "validating XML parsers" check whether $a \models \varphi$). The notion of constraint language provides a unifying perspective and a basis for comparing formalisms like DTD, XML-SCHEMA, RELAX, RDF-SCHEMA, wrt. their expressiveness and complexity.

Definition 2 (Subsumption) We say that $\mathcal{C}'$ *subsumes* $\mathcal{C}$ *wrt.* $\mathcal{A}$, denoted $\mathcal{C}' \succ \mathcal{C}$, if for all $\varphi \in \mathcal{C}$ there is a $enc(\varphi) \in \mathcal{C}'$ s.t. for all $a \in \mathcal{A}$: $a \models \varphi$ iff $a \models enc(\varphi)$. □

As a constraint language, DTD can express only certain structural constraints over XML, all of which have equivalent encodings in XML-SCHEMA. Hence XML-SCHEMA subsumes DTD. On the other hand, XML-SCHEMA is a much more complex formalism than DTD, so a more complex validator is needed when reinstantiating the archive, thereby actually increasing the infrastructure dependence (at least for archives where DTD constraints are sufficient). To overcome this problem, we propose to use a generic, universal formalism that allows one to *specify and execute* other constraint languages:

4.2 Self-Validating Archives

Example 1 (Logic DTD Validator) Consider the following F-LOGIC rules [7]:

```
%%% Rules for ⟨!ELEMENT X (Y,Z)⟩
(1) false ← P:X, not (P.1):Y.
(2) false ← P:X, not (P.2):Z.
(3) false ← P:X, not P[_→_].
(4) false ← P:X[N→_], not N=1, not N=2.
%%% Rules for ⟨!ELEMENT X (Y | Z)⟩
(5) false ← P:X[1→A], not A:Y, not A:Z
(6) false ← P:X, not P[_→_].
(7) false ← P:X[N→_], not N=1.
%%% Rule for ⟨!ELEMENT X (Y)*⟩
(8) false ← P:X[_→C], not C:Y.
```

The rule templates show how to generate for each $\varphi \in$ DTD a logic program $enc(\varphi)$ in F-LOGIC, which derives false iff a given document $a \in$ XML is not valid wrt. φ: *e.g.*, if the first child is not Y (1), or if there are more than two children (4). □

The previous logical DTD specification does not involve recursion and can be expressed in classical first-order logic FO. However, for expressing transitive constraints (*e.g.*, subclassing, value inheritance, *etc.*) fixpoint extensions to FO (like DATALOG or F-LOGIC) are necessary.

Proposition 1 *(i)* XML-SCHEMA $\succ$ DTD, *(ii)* F-LOGIC $\succ$ DTD. □

[9] $\approx$ DTDs + datatypes + type extensions/restrictions + ...

[10] $\approx$ DTDs + (datatypes, ancestor-sensitive content models, local scoping, ...) – (entities, notations) ...

Note that there is a subtle but important difference between the two subsumptions: In order to "recover" the original DTD constraint via *(i)*, one needs to understand the specific XML-SCHEMA standard, and in order to execute (*i.e.*, check) the constraint, one needs a *specific* XML-SCHEMA validator. In contrast, the subsumption of *(ii)* as sketched above contains its own *declarative, executable specification*, hence is *self-contained* and *infrastructure independent*. In this case, *i.e.*, if an AIP contains (in KP) an *executable specification* of the constraint φ, we speak of a *self-validating archive*. This means that at dissemination time we only need a single *logic virtual machine* (*e.g.*, to execute PROLOG or F-LOGIC) on which we can run all logically defined constraints. The generic engine for executing "foreign constraints" does not have to be a logic one though: *e.g.*, a RELAX validator has been written in XSLT [18]. Then, at reinstantiation time, one only needs a generic XSLT engine for checking RELAX constraints.[11]

4.3 Self-Instantiating Archives

A self-validating archive captures one or more snapshots of the archived collection at certain stages during the ingestion process, together with constraints φ for each snapshot. The notion of *self-instantiating archive* goes a step further and aims at archiving also the *transformations* of the ingestion network themselves. Thus, instead of adding only descriptive metadata about a transformation which is external to the archive, we include the "transformation knowledge" thereby internalizing complete parts of the ingestion process.

As before, we can maximize infrastructure independence by employing a universal formalism whose specifications can be executed on a virtual (logic or XML-based) engine – ideally the same one as used for checking constraints. To do so, we model an ingestion network as a graph of *database transformations*. This is a natural assumption for most real transformations (apart form very low level reformatting and conversion steps).

Definition 3 (Ingestion Network) Let $\mathcal{T}$ be a set of transformations $t : \mathcal{A} \to \mathcal{A}$, and $\mathcal{S}$ a set of *stages*. An *ingestion network* $\mathcal{IN}$ is a finite set of labeled edges $s \to_t s'$, having associated *preconditions* $\varphi(s)$ and *postconditions* $\varphi(s')$, for $s, s' \in \mathcal{S}, t \in \mathcal{T}, \varphi(s), \varphi(s') \in \mathcal{C}$. □

[11] However, in the archival context, instead of employing the latest, rapidly changing formalisms, a "timeless" logical approach may be preferable.

We call the edges of $\mathcal{IN}$ *pipes* and say that an archive $a \in \mathcal{A}$ is *acceptable for* ("may pass through") the pipe $s \to_t s'$, if $a \models \varphi(s)$ and $t(a) \models \varphi(s')$. Since $\mathcal{IN}$ can have loops, fixpoint or closure operations can be handled. If there are multiple t-edges $s \to_t s'_i$ outgoing from s, then one s'_0 is distinguished to identify the *main pipe* $s \to_t s'_0$; the remaining $s \to_t s'_i$ are called *contingency pipes*. The idea is that the postcondition $\varphi(s'_0)$ captures the normal, desired case for applying t at s, whereas the other $\varphi(s'_i)$ handle exceptions and errors. In particular, for $\varphi(s'_1) = \neg\, \varphi(s'_0)$ we catch *all* archives that fail the main pipe at s, so s'_1 can be used to abort the ingestion and report the integrity violation $\neg\, \varphi(s'_0)$. Alternatively, s'_1 may have further outgoing contingency pipes aimed at rectifying the problem.

When an archive a successfully passes through the ingestion net, one or more of the transformed versions a' are archived. One benefit of archiving the transformations of the pipeline (SIP $\to_{t_1} \cdots \to_{t_n}$ AIP) in an infrastructure independent way is that knowledge, that was available at ingestion time and is possibly hidden within the transformation, is preserved. Moreover, some of the transformations yield user-views (AIP $\to_{t_1} \cdots \to_{t_m}$ DIP), *e.g.*, topic maps or HTML pages. By archiving self-contained, executable specifications of these mappings, the archival reinstantiation process can be automated to a large extent using infrastructure independent representations.

Properties of Transformations. By modeling the ingestion net as a graph of database mappings, we can formally study properties of the ingestion process, *e.g.*, the data complexity of a transformation. Based on the time complexity and resource costs of transformations, we can decide whether a collection should be recreated on demand via the ingestion net, or whether it is preferable to materialize and later retrieve snapshots of the collection. This choice is common in computer science even outside databases, *e.g.*, for scientific computations in virtual data grids, applications can choose to retrieve previously stored results from disk or recompute the data product, possibly using stored partial results [8]. Note that invertible transformations of an ingestion net are content preserving. For transformations t that are not specific to a collection, it can be worthwhile to derive and implement the inverse mapping t^{-1} thereby guaranteeing that t is content preserving.

Example 2 (Inverse Wrapper) Consider a document collection $\{a_1, a_2, \ldots\} \subseteq$ XHTML for which a common wrapper t has been provided s.t. $t(a_i) = a'_i \in$ XML. The exact inverse mapping may be impracticable to construct, but a "reasonably equivalent" t^{-1} (*i.e.*, modulo whitespaces, irrelevant formatting details, *etc.*) may be easy to define as an XSLT stylesheet. Thus, the output of the pipe $a_i \rightarrow_t a'_i \rightarrow_{t^{-1}} a''_i \in$ XHTML can be seen as a *normalized* (X)HTML version of the input a_i. By restricting to normalized input, t becomes invertible, and the XSLT script acts as an "inverse wrapper" for presenting the collection. □

5 Case Study: The Senate Collection

In a research collaboration with the National Archives and Records Administration (NARA), SDSC developed an information management architecture and prototype for digital archives. Below, we illustrate some of the aspects of our archival architecture, using the *Senate Legislative Activities* collection (SLA), one of the reference collections that NARA provided for research purposes.

Collection Submission and Initial Model. The SLA collection contains an extract of the 106th Congress database *bills*, *amendments*, and *resolutions* (short: BARs). SLA was physically submitted on CD-ROM as 99 files in Microsoft's *Rich Text Format* (RTF), one per *active* senator, and organized to reflect a particular senator's legislative contribution over the course of the 106th Congress. Based on a visual inspection of the files, an *initial conceptual model* CM_0 with the following structure was assumed:

- **Header section**: includes the *senator name* (*e.g.*, "Paul S. Sarbanes"), *state* ("Maryland"), *reporting period* ("January 06, 1999 to March 31, 2000"), and *reporting entity* ("Senate Computer Center Office of the Sergeant at Arms and Committee on Rules and Administration")
- **Section I**: *Sponsored Measures*, **Section II**: *Cosponsored Measures*, **Section III**: *Sponsored Measures Grouped by Committee Referral*, **Section IV**: *Cosponsored Measures Organized by Committee Referral*, **Section V**: *Sponsored Amendments*, **Section VI**: *Cosponsored Amendments*,
- **Section VII**: *Subject Index to Sponsored and Cosponsored Measures and Amendments*.

CM_0 also modeled the fact that Sections III and IV contain the same bills and amendments as Sections I and II, but *grouped by committee referral* (*e.g.*, "Senate Armed Services" and "House Judiciary"), and that Section VII contains a list of subjects with references to corresponding BAR indentifiers: "Zoning and zoning law → S.9, S.Con.Res.10, S.Res.41, S.J.Res.39". *Measures* are *bills* and *resolutions*; the latter have three subtypes: *simple*, *joint*, and *concurrent*.

Finally, CM_0 identified 14 initial *data fields* DF_0 (=attributes) that needed to be extracted.[12]

Ingestion Process. Figure 2 depicts the ingestion network as it eventually evolved: The (presumed) conversion from (MS Word) DOC to RTF happened outside of the ingestion net, since the accessioning policy prescribed SIPs in RTF format.

- $S_1 \rightarrow S_2$:[13] A first, supposedly content preserving, conversion to HTML using MS Word turned out to be lossy when checked against CM_0: the groupings in Sections III and IV were no longer part of the HTML files,[14] so it was impossible to associate a measure with a committee!
- $S_1 \rightarrow S_3$: the conversion from RTF to an information preserving XML representation was accomplished using an rtf2xml module[15] for OmniMark, a stream-oriented rule-based data extraction and programming language.
- $S_3 \rightarrow S_4$: this main wrapping step was used to extract data according to the initial data fields DF_0. In order to simplify the (Perl) wrapper module and make it more generic, we used a *flat, occurrence-based* representation for data extraction: each data field (attribute) was recorded in OAV form, *i.e.*,
- (*occurrence*, *attribute*, *value*)

The occurrence has to be fine-grained enough for the transformation to be information preserving (in our case *occurrence* = (*filename*, *line-number*)). The *scope of an occurrence* is that part of the linearized document which defines the extent of the occurrence. For example, in case of an occurrence based on line numbers, the scope is from the first character of the line to the last character of the line. In case of XML, the scope of an occurrence may often be associated with element boundaries (but finer occurrence granules

[12] *abstract, bar_id, committee, congressional_record, cosponsors, date_introduced, digest, latest_status, official_title, sponsor, statement_of_purpose, status_actions, submitted_by, submitted_for*

[13] this dead end is only an example for existing pitfalls; $S_1 \rightarrow S_2$ is *not* archived.

[14] this crucial information was part of the RTF *page header* but left no trace whatsoever in the HTML

[15] from Rick Geimer at `xmeta.com`

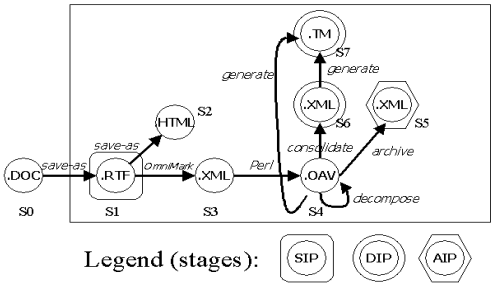

Figure 2. Ingestion Network: Senate Collection

may be defined for XML as well). By employing the "deconstructing" OAV model, the wrapper program could be designed simpler, more modular and thus easier to reuse: *e.g.*, date_introduced could show up in the file of Senator Paul Sarbanes (senator_id=106) at line 25 with value 01/19/1999 and also in line 106 at line 55 with value 03/15/2000. This information is recorded with two tuples: ((106,25), 'date_introduced', '01/19/1999') and ((106,55), 'date_introduced', '03/15/2000').

• $S_4 \rightarrow S_4$: some candidate attributes from DF_0 had to be decomposed further, which is modeled by a recursive *closure step* $S_4 \rightarrow S_4$, corresponding to a sequence $DF_1, \ldots, DF_n$ of refinements of the data-fields, *e.g.*, DF_1: list_of_sponsors $\rightarrow$ [sponsor], and DF_2: sponsor$\rightarrow$ (name, date).

• $S_4 \rightarrow S_5$: this "reconstructing" step builds the desired archival information packages AIP in XML. Content *and structure* of the original SIPs is preserved by reassembling larger objects from subobjects using their occurrence values. From the created XML AIPs, DTDs like the following can be inferred (and included as a constraint φ in KP):

```
<!ELEMENT SLA_collection
       (senate_file*)>
<!ELEMENT senate_file
       (file_name, header_page?,
       section*, subject_index?)>
<!ELEMENT section
       (sec_number, sec_name, bar*)>
<!ELEMENT bar
       (bill | amendment | resolution)>
...
```

• $S_4 \rightarrow S_6$: this conceptual-level transformation creates a *consolidated version* from the collection. For example, SLA contains 44,145 *occurrences* of BARs, however there are only 5,632 *distinct* BAR objects. (Alternatively, this version could have been derived from S_5.) This step can be seen as a reverse-engineering of the original database content, of which SLA is only a view (group BARs by senator, for each senator group by measures, committee, *etc.*)

As part of the consolidation transformation, it is natural to perform conceptual-level integrity checks: *e.g.*, at this level it is easy to define a constraint φ that checks for *completeness* of the collection (*i.e.*, if each senator occurring somewhere in the collection also has a corresponding senator file – a simple declarative query reveals the answer: no!). Note that a consolidated version provides an *additional* archival service; but it is mandatory to also preserve a non-consolidated "raw version" (*e.g.*, as derived from the OAV model).

• $S_4, S_6 \rightarrow S_7$: these transformations create a *topic map* version and thus provide additional conceptual-level "hooks" into the consolidated and OAV version.

6 Conclusions

We have presented a framework for the preservation of digital data, based on a forward migration approach using XML as the common archival format and generic XML tools (XML wrappers, query and transformation engines). The approach has been extended towards self-validating knowledge-based archives: *self-validating* means that declarative constraints about the collection are included in *executable form* (as logic rules). Most parts of the ingestion network (apart from S_7 which is under development) have been implemented for a concrete collection. Note that all transformations following the OAV format can be very naturally expressed in a high-level object-oriented logic language (*e.g.*, F-LOGIC). By including the corresponding rules as part of the archive, a *self-instantiating*, self-validating archive can be constructed.

References

[1] C. Baru, V. Chu, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-Based Information Mediation for Digital Libraries. In *ACM Conf. on Digital Libraries (DL)*, pages 214–215, Berkeley, CA, 1999. exhibition program.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[3] Conceptual Graph Standard. dpANS, NCITS.T2/98-003, http://www.bestweb.net/~sowa/cg/cgdpans.htm, Aug. 1999.

[4] Encoded Archival Description (EAD). http://lcweb.loc.gov/ead/, June 1998.

[5] J. Garrett and D. Waters, editors. *Preserving Digital Information – Report of the Task Force on Archiving of Digital Information*, May 1996. http://www.rlg.org/ArchTF/.

[6] Knowledge Interchange Format (KIF). dpANS, NCITS.T2/98-004, http://logic.stanford.edu/kif/, 1999.

[7] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[8] R. Moore. Knowledge-Based Grids. In *18th IEEE Symposium on Mass Storage Systems*, San Diego, 2001.

[9] R. Moore, C. Baru, A. Rajasekar, B. Ludäscher, R. Marciano, M. Wan, W. Schroeder, and A. Gupta. Collection-Based Persistent Digital Archives. *D-Lib Magazine*, 6(3,4), 2000.

[10] M. Murata. RELAX (REgular LAnguage description for XML). http://www.xml.gr.jp/relax/, Oct. 2000.

[11] Reference Model for an Open Archival Information System (OAIS). submitted as ISO draft, http://www.ccsds.org/documents/pdf/CCSDS-650.0-R-1.pdf, 1999.

[12] A. Paepcke, R. Brandriff, G. Janee, R. Larson, B. Ludäscher, S. Melnik, and S. Raghavan. Search Middleware and the Simple Digital Library Interoperability Protocol. *D-Lib Magazine*, 6(3), 2000.

[13] Resource Description Framework (RDF). W3C Recommendation www.w3.org/TR/REC-rdf-syntax, Feb. 1999.

[14] ISO/IEC FCD 13250 – Topic Maps, 1999. http://www.ornl.gov/sgml/sc34/document/0058.htm.

[15] OMG XML Metadata Interchange (XMI). www.omg.org/cgi-bin/doc?ad/99-10-02, 1999.

[16] Extensible Markup Language (XML). www.w3.org/XML/, 1998.

[17] XML Schema, Working Draft. www.w3.org/TR/xmlschema-{1,2}, Sept. 2000.

[18] K. Yonekura. RELAX Verifier for XSLT. http://www.geocities.co.jp/SiliconValley-Bay/4639/, Oct. 2000.