PAWN: a Policy-Driven Software Environment for Implementing Producer-Archive Interactions in Support of Long Term Digital Preservation

Mike Smorul, Mike McGann, Joseph JaJa; Institute for Advanced Computer Studies, University of Maryland, College Park

Abstract

The ingestion process of digital objects into a long term archive constitutes a critical phase of the overall archiving process during which the object's content, metadata, context, and provenance have to be assembled correctly. This task becomes quite complicated when there are many independent producers involved, each with a possibly different arrangement with the archive. In this paper, we describe the underpinnings of a novel software environment for capturing the interactions between distributed producers and an archive which ensures the inclusion of all the necessary elements to preserve the digital information. A prototype system called PAWN (Producer – Archive Workflow Network) provides a flexible and scalable platform for creating and securely ingesting digital information into a remote archive while allowing flexible interactions between the producer and the archive. PAWN is policy – driven with built-in core functions and policies that can be customized to address ingestion requirements for any archiving community. The environment is platform – independent and is based on open standards and web technologies, and is designed to operate across multiple administrative domains using strong security mechanisms. The latest PAWN release version .5 is currently under testing by a number of projects that involve realistic environments with significant amounts of digital data to be preserved.

1. Introduction

A large portion of the scientific, business, cultural, and government digital information being created today needs to be archived and preserved for future use of periods ranging from a few years to decades and sometimes centuries. Since the mid nineties, the issue of long-term preservation of digital information has received considerable attention by major archiving communities, library organizations, government agencies, scientific communities, and individual researchers. These efforts (such as [1,2,3]) have identified major challenges regarding the technology infrastructure needed to achieve long-term preservation of and access to digital information. These challenges include the handling of technology evolution in computer hardware and media, systems and applications software; the maintenance of the authenticity and integrity of the data throughout its lifetime; and risk management and disaster recovery due to technology degradation and failure, natural disasters, operational errors, and malicious corruption of the data. Clearly, the long-term preservation of digital information is a process that must begin before the data is ingested into an archival system and that must remain continuously active throughout the lifecycle management of the digital objects.

In this paper, we describe PAWN – Producer Archive Workflow Network – a software system that provides a flexible and scalable platform for creating and reliably ingesting digital information into a remote archive by a wide variety of producers while allowing the customization of the interactions between each producer and the archive. PAWN builds on the previous version [4] by offering a considerably more flexible environment including policy driven management of the ingestion process. More specifically, PAWN presents:

- A flexible and robust environment for defining and implementing interaction policies between producers and archives.
- A secure, reliable, and scalable ingestion from distributed producers into an archive, which ensures the inclusion of all the necessary information to preserve each digital object.
- Built-in core functions and policies that can be customized to address ingestion pipeline requirements for any archiving community.
- A platform-independent system based on open standards and web technologies, which is designed to operate across multiple administrative domains using, in its current version, PKI and SAML assertions.

PAWN enables the interactions between the producers and the archive within a secure, reliable, and scalable environment. The term producer [5,6] designates the persons or systems which supply the archive with the information to be preserved. Note that the archive itself can either be centralized or distributed. In fact, our PAWN testing has been performed on the pilot persistent archive consisting of a federated data grid that includes storage systems at the San Diego Supercomputer Center, the National Archives, and the University of Maryland.

PAWN encapsulates properties of content, structure, context, and presentation within a digital object architecture making use of METS (Metadata Encoding and Transmission Standard) to define the ingest package [7,8]. We adopt the framework developed by the Open Archival Information Systems (OAIS) [5] focusing on the Producer – Archive interactions [6], in which producers prepare and transfer the information to be preserved to an archive, which is responsible for managing the digital information and for providing an interface to the consumers (data users). For each stage, OAIS provides a detailed model of the information, called respectively the Submission Information Package (SIP), the Archival Information Package (AIP), and the Dissemination Information Package (DIP).

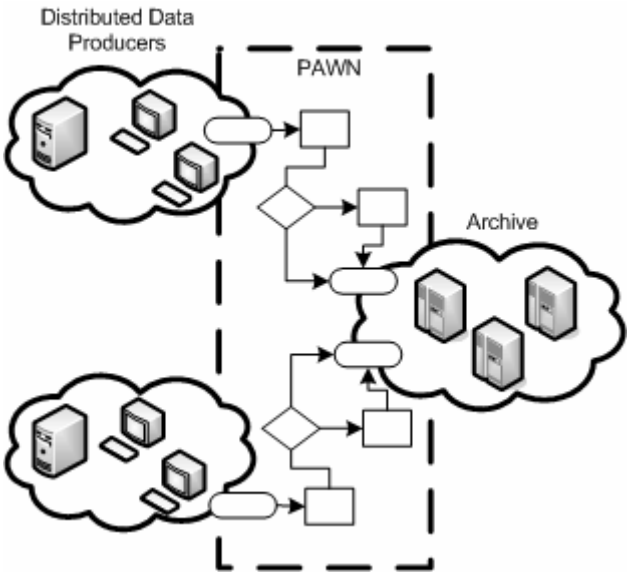

Figure 1: Overall PAWN Environment

Most relevant to PAWN is the SIP that consists of the following components (see [5] for more details):

- First is the *Content Information* (CI), which is divided into two parts.
    - *Content Data Object*, consisting of the actual bitstream to be preserved.
    - *Representation Information*, which includes file format, endian issues, and encoding format. Consider for example the case of image files. The corresponding information would be given about expected header formats, location of internal checksums and what utilities can be used to verify the file can be loaded. Enough information should be given to ensure that the archive would be able to process the bitstream.
- Second is *Preservation Description Information* (PDI) that contains four parts.
    - *Chain of custody*.
    - *Context* in relation to other Information Packages
    - *Reference information* unique to the bitstream (eg, ISBN, global identifier, etc)
    - *Fixity information* required to ensure bitstream integrity (eg, hashes, or checksums).
- *Packaging Information* describes the relationship between CI and PDI. This describes the physical location of the Content Information and corresponding PDI.
- *Descriptive Information* used for data discovery. This user-defined metadata will be supplied during the ingestion of the bitstream at the producer. This includes descriptions of the bitstream, authorship and other elements (e.g. Dublin Core).

PAWN assembles a SIP and uses METS as the default method to represent the various elements of the SIP. However PAWN can also publish into other packaging formats such as XFDU [9] that was developed for handling scientific data.

In the next section, we present the core concepts underlying the design and architecture of PAWN, while Section 3 is devoted to an overview of the PAWN architecture and its corresponding software components.

2. Basic Concepts

PAWN supports an environment in which many distributed producers independently of each other manage their ingestion strategy, as well as, independently assemble and transfer their data to the archive. In fact, PAWN provides a common infrastructure for both producers and the archive to manage and monitor the overall ingestion process. As articulated in [6], it is expected that negotiations between the producer and the archive about the details of what needs to be preserved, types of data objects, transfer conditions, user access policy, and delivery schedule must take place prior to any data transfer from a producer to the archive. These negotiations should produce a clear understanding of the elements necessary to assemble the corresponding SIP for each item to be preserved, referred to as a *submission agreement*. In particular, information related to data representation, context, chain of custody (including date of transfer), preservation and access has to be agreed upon prior to ingestion. PAWN encapsulates such information into a document called *a record schedule*, which gets embedded within the PAWN environment as we will see later.

In order to manage distributed ingestion by independent producers, the archive organizes the producers into some kind of a global structure (typically a hierarchy), which provides an overall context into which transferred data can be linked to. Using this structure, we further group the producers into *domains*, each domain to be viewed as a single logical entity by the archive. Within each domain, we attach an overall hierarchy that broadly lists all the types of data produced by this domain; a record schedule as defined above; and customized subsets of the record schedule, called *record sets*, which serve as submission templates for producers within this domain. In particular, a record set is a convenient grouping of items from the record schedule and serves as a template to be filled by an end-user.

In summary, each producer operates within a domain and can create SIP packages and submit them directly to the archive using the record sets associated with his/her account. Our framework allows a separate data organization per domain, and moreover the final destination of the data at the archive may also be assigned on a per domain basis.

We next provide more details about domains, policy management and role assignment, creation and management of SIP packages, and the security infrastructure, which constitute the underpinnings of the PAWN distributed environment.

Domains and their Structures

A domain in PAWN corresponds to a group of individual producers that share a common agreement with the archive. Domains will typically be established along administrative boundaries. Consider for example the process of setting up an archive for an academic institution, structured administratively around the Offices of the President and the Vice Presidents, and the Colleges within which all the departments and centers operate.

Various members of these administrative units constitute potential producers of different types of data to be archived for various lengths of time. A possible organization of the producers consists of a domain for the Offices of the President and Vice Presidents, and a domain for each of the colleges in the institution. A record schedule for a college domain for example will include the preservation information and actions required for each type of data produced by the college (including all the units within the college) such as administrative, financial, publication reports, and so on. PAWN enables the customization of the record schedule to each individual producer within a domain (e.g. a faculty member, an administrator, a member of the administrative or technical staff) so that only the relevant pieces of the record schedule will be seen by that producer. In fact, PAWN presents each producer with a list of relevant record sets (customized from the record schedule) that she must fill. For example, a faculty member will be presented with templates corresponding to technical reports, conference/journal papers, presentations or posters, which need to be filled, after which the data can be attached and transferred to the archive. PAWN will automatically extract all the necessary items for the SIP, assemble and transfer the corresponding SIP to the archive. Figure 2 illustrates a possible organization of the domains associated with an academic institution.

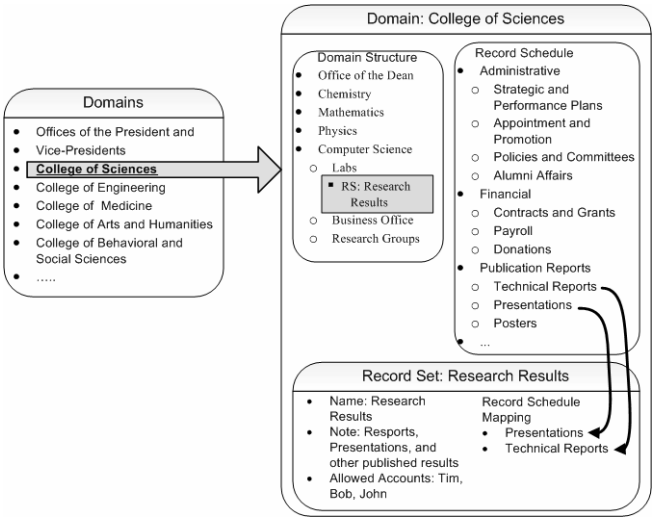

Figure 2 Possible Organization of Domains Associated with an Archive for an Academic Institution

Record sets provide a convenient packaging of contextual information for creating a SIP, and considerably simplify package creation by presenting the end-user with a simple template to fill. A Record set typically contains a descriptive name, list of allowed users, a mapping into the record schedule, and a note regarding the set's use. The mapping from a record schedule into a record set also allows managers to limit what types of documents an end-user may ingest.

Policy Management and Roles in PAWN

Given that the interactions between the producer and the archive can vary significantly depending on the communities and organizations involved, PAWN provides a flexible environment to enable the customization of the ingestion process to capture a wide variety of possible interactions between the producer and the archive. This is done through the introduction of *roles*. Each account in PAWN is assigned a role. A role is defined by a group of actions allowed. These actions range from package management (view, modify, delete items) to operations that are used for account management, record set manipulation and record set creation. Roles are configurable and can be created as needed depending on the relationship between the producer and archive.

By default, there are four preconfigured roles in PAWN. These are a *global administrator* (GA), *records manager* (RM), *archive manager* (AM), and *producer* (P). The global administrator is able to perform all actions including the creation, modification, and deletion of domains, and setting up manager accounts. The records manager is expected to sit within the data producers administrative structure and is able to create record sets, end-user accounts, and assist in creating and editing packages. The archive manager can assist in managing record organization, edit submitted packages, move items from packages into long term storage, and remove items from PAWN after processing. The last role is an end-user (data producer) who creates and submits packages to PAWN for preservation.

Creation and Management of Packages

Information in PAWN consists of producer submitted packages. A package contains data and metadata that a producer wishes to archive. Combined with additional contextual information extracted by PAWN, a package contains all necessary elements to assemble a SIP. In fact, a package is constructed by attaching data to the categories of a record set. This data is organized into a hierarchy rooted at the record set category with data and metadata attached at various points in the hierarchy defined by the record set.

Within a package, the content information (as defined by OAIS) is provided by the physical bits attached and the representation information is supplied by the client as a mime type. A verification service such as the format verification service available through FOCUS [10] can be used at the archive's staging area to validate the representation information.

PAWN provides for Preservation Description Information (as defined by OAIS) as follows. The chain of custody is recorded in two places. The package tracks the original location of data on a client's computer and the identifier and location of the data on a PAWN receiving server. Second, an event log described below tracks all actions and final destination of data as it moves though PAWN. Context of a package is provided by the record set structure. The record set hierarchy is used to show the location and use of packages within a larger organization. PAWN and its archival resources track reference information for all data that moves through PAWN using internal identifiers. Fixity information is provided by clients in the form of SHA-256 digests on all items. In addition, PAWN will check all data and metadata for consistency.

Packaging information in PAWN is described by using METS files to track data objects and all associated system and administrative metadata. The METS files record the hierarchical structure of the package and where any descriptive metadata may be associated in that hierarchy.

Packages in PAWN follow a simple lifecycle from creation through the final stage consisting of publication into the archive

and removal from PAWN. All actions, except for details of the initial package creation, are logged. Once a package leaves PAWN and its content removed, the log of the package will remain.

The typical workflow for items in PAWN is the following:

1. Producer selects a record set to use as a package template.
2. PAWN builds a package locally and transfers it to a receiving server. A package does not have to be compiled at once, but can be appended and modified later as necessary.
3. A producer may lock the package to prevent further modification and signal that the submission is complete.
4. A manager at either the producer or the archive may review the package and optionally reject submitted items. If an item is rejected, the package can be unlocked, modified, and resubmitted by the producer.
5. Submitted and locked packages that have not been rejected can be passed to the archive. Final destination and any errors from the archive are logged.
6. After archiving, a manager at the archive can review the package log and remove the package from PAWN if it is no longer needed. The package log remains.

The log for a package in PAWN tracks all changes after initial creation. It records the type of action, who performed the action, and any errors that may have occurred. In addition, the log also tracks specific items that may have been affected by the action.

The following package-level events are logged:

- Lock – Package is locked and no further modification is allowed.
- Unlock – Package is made available for modification.
- Finish / Remove – Entire package is removed and log file is stored.

The following events occur to individual items, where the affected items are also logged:

- Reject – Individual items are rejected.
- Accept – Items are no longer rejected and may be archived.
- Archive – Items are pushed to an archival resource. Final destination is logged.
- Add / Remove – Items are added or removed from a package.

Security Model

Responsibility for security in PAWN is distributed between the archive receiving components and management server(s). This requires that a certain level of trust exist between components on the producer side, and components on the archive side. The trust exists only between producer and archive, not between various producer installations.

From a trust perspective, there are three types of calls that are made in PAWN. First are producer only calls where a locally authenticated client communicates with its management server. Second are package management calls where a producer-authenticated client needs to access package data stored at an archive receiving server. Third are management calls between components at the archive. The package management calls require a trust relationship between the producer and archive.

We use WS-Security with Security Assertion Markup Languge(SAML)[11] to provide the necessary security for calls that cross administrative boundaries. The Apache WSS4J project provides the physical mechanism for signing web service calls. The following steps show how trust is established between a producer and the archive.

1. Producer management server gets a signed key pair that will be used to create SAML assertions.
2. A copy of the producer's public key and SAML namespace is transferred to archive. (Pre-existing trust relationship, locked briefcase, etc.)
3. All archive components receive a copy of the key and namespace
4. Client authenticates to its local management server and presents a public key.
5. Producer inspects supplied key, ensures it's signed by a recognized CA and creates a signed SAML assertion with role information and client's public key embedded. The assertion is returned to the client.
6. Client inspects the resulting assertion to ensure that it was signed by its producer.
7. Client creates a web service call, signs it with its private key and embeds the SAML assertion in the call header.
8. Archive service receives assertion, checks to ensure that call signature matches embedded key and checks assertion signature against local copy of producer's key and namespace.
9. Embedded role information is used to finally determine if

```
   <Event Date="2006-05-31T11:06:51.133-04:00" Domain="umiacs"
Issuer="http://umiacs.umd.edu" User="umiacs:toaster">
      <Reject/>
      <ItemList ManifestId="urn:pawn:mets-id.1148670951996293000"
      ManifestIncluded="false" Name="backgrounds" Obligation="z1">

            <Parents>urn:pawn:mets-id.1148670947787630000</Parents>
            <Item ID="urn:pawn:file-id.xccfcf684cec05554516b9a2b6b14c2a3"
      Name="utopy.jpg"/>

      </ItemList>
   </Event>
```

Figure 3 Sample log example

authorized to use call.

3. PAWN Architecture and Main Software Components

The PAWN environment is built upon four major software components: management server, client, scheduler, and receiving server. Briefly, the management server builds and tracks accounts, record schedules, record sets, package lists, and provides security mechanisms for the domains. The client is used to ingest data, manage users and data organization, and to trigger transfers into the archive. The scheduler allocates resources on the receiving servers and manages all the security services of the receiving servers. The receiving server stores and manages data transferred from clients, including the invocation of validation services on the data received.

These components are shown in Figure 4. It should be noted that there may be multiple sets of management servers and clients connecting to one scheduler and a set of receiving servers.

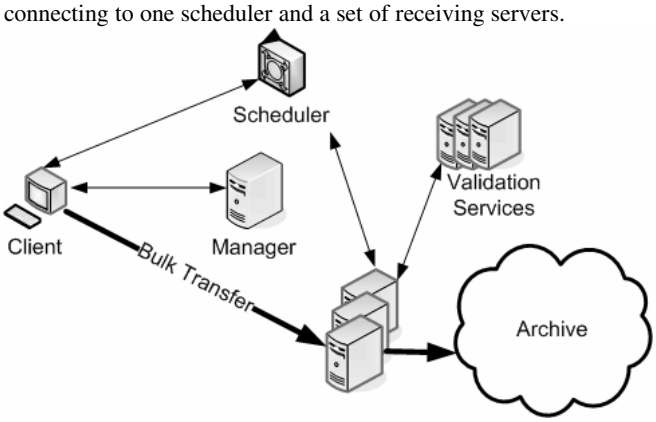

Figure 4: PAWN Components

Management Server

The PAWN management server manages accounts and domains. In PAWN, there may be multiple management servers whose clients submit data to resources sitting at an archive. Management servers are independent, relying only on the archive. A management server does not share accounts and record information with other management servers.

When a client connects to PAWN, it authenticates to a management server. The server determines what permission the client has been allocated and returns it to the client. The client uses these permissions to present the appropriate interface to the end user.

The management server tracks all packages that have been submitted to the archive through any of the domains under its management. This includes packages that may have been removed from PAWN as well. The management server tracks the current state of a package, whether it is locked, unlocked, or has been removed. When a package is removed from PAWN, a copy of its log file is stored on the management server to provide a record of what actions may have occurred on items in the package.

In a security context, each management server is responsible for issuing SAML assertions for all of its clients. Clients send a request for a SAML token during login and upon successful login, and presentation of a valid public key, the server will issue the assertion. The issued assertions are signed with a key pair that is trusted by the archive. Each issued assertion contains the list of roles that a particular client is allowed to perform.

Scheduler

The scheduler in PAWN provides a gateway into archive based services. It serves two roles. First it allocates resources on receiving servers for client packages and second it acts as a repository for receiving server configurations. The scheduling aspect is handled using the Condor classad[12] system and an interface is provided to all configurations on the scheduler.

Scheduling in PAWN is performed using the condor classad library. This library helps create and evaluate classads. Classads are mappings from attribute names to expressions. There is a protocol for evaluating one classad with respect to another in order to match compatible classads[12]. In PAWN, classads are constructed to represent client requirements and receiving server resources. The classads are then evaluated to find compatible matches, and in the case of multiple matches the most desirable match. The classads are based primarily on required disk space of client packages and available receiving server space.

Periodically, each receiving server in PAWN will publish a classad to the scheduler describing currently used and available resources. The receiving server will also specify requirements for accepting client submissions. Any receiving server that fails to publish a classad within a specified time is removed from the list of available servers.

When a client requests a resource, its request is transformed into a classad that specifies information about the current client and the requirements are set to the required space. This request flow is shown in Figure 5.

The requirements from both parties can be expanded in the future. For example, a client may wish to request receiving servers that contain specific archival services, or a receiving server may wish to restrict access to clients from a given domain.

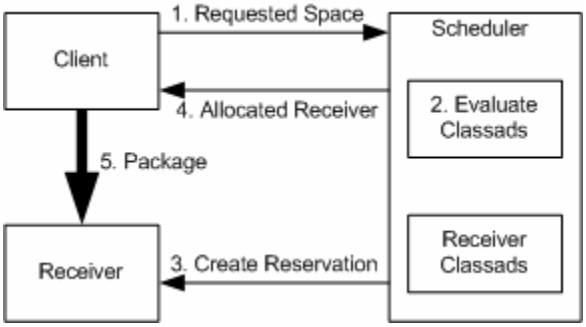

Figure 5: Scheduler Workflow

Receiving Server

A receiving server in PAWN provides storage space for packages that are under the control of PAWN. It enforces access control on items in a package, logs actions that occur to a package, and handles publishing packages into archival resources.

A receiving server consists of one or more storage pools that are used to house packages. The states of these pools are

periodically published to the scheduler in the form of a classad. These classads allow the scheduler to place new packages in a particular storage pool on a receiver.

Each package in pool contains its own set of access controls. These access controls allow packages from various domains and management server to be stored in the same pool. Packages also store the state of all contained items and limit actions depending on the state of an item. For example, a package will not allow an archive attempt on an item that has been rejected.

When packages are removed from the receiving server, all items regarding the package are removed, and a copy of the package log is sent to the management server.

Receiving servers also contain the necessary software to transfer packages to a long term digital archive. PAWN provides a simple API that allows 3rd party developers to create gateways into their archive. It is recognized that an archive may have multiple repositories to store various types of data. As such, PAWN allows any number of gateways to be defined on a per-domain basis. This allows differing policy decisions regarding the final placement of objects within PAWN to be represented.

Client Interface

The PAWN client provides a workbench for accessing most functionality in PAWN. It is used to configure domains, accounts, record organizational structures and manage packages.

The client provides a mechanism for developing custom package builders. PAWN provides a simple data model that custom package builders can use to create packages. The package builder supplies an interface to the end user for package creation and later modification.

4. Conclusion

We have presented an overview of the basic concepts and architecture for the PAWN environment. We have argued that PAWN presents an extremely flexible environment to capture a wide variety of possible interactions between distributed producers and an archive. Moreover, the design of PAWN paid from the beginning a particular attention to security, reliability, and scalability using open standards and web technologies.

5. Acknowledgements

This work was supported in part by the National Archives and Records Administration, ERA Program through the National Science Foundation.

6. References

[1] Trusted Digital Repositories: Attributes and Responsibilities, Report by the Research Libraries Group, Mountain View, CA, May 2002. http://www.rlg.org

[2] It's About Time: Research and Challenges in Digital Archiving and Long-Term Preservation, Report on Workshop on Research Challenges in Digital Archiving: Toward a National Infrastructure for Long-Term Preservation of Digital Information, 2002. http://www.loc.gov/

[3] The National Archives and Records Administration: Center for Electronic Records. http://www.nara.gov/nara/electronic/

[4] PAWN: Producer – Archive Workflow Network in Support of Digital Preservation, M. Smorul, J. JaJa, Y. Wang, and F. McCall, UMIACS Technical Report, UMIACS-TR-2004-49, University of Maryland, College Park, 2004.

[5] Reference Model for an Open Archival Information System (OAIS), CCSDS 650.0-B-1, Blue Book, Issue 1, January 2002 [Equivalent to ISO 14721:2002].

[6] Producer – Archive Interface Methodology: Abstract Standard, Consultative Committee for Space Data Systems, CCSDS-651.0-R-1, Red Book, December 2002.

[7] METS – Metadata Encoding and Transmission Standard http://www.loc.gov/standards/mets/

[8] METS Profile 1.0 Requirements, J. McDonough. http://www.loc.gov/standards/mets/profile_docs/METS.profile.requirements.rtf, June 2003.

[9] XML Formatted Data Unit – XFDU: http://sindbad.gsfc.nasa.gov/xfdu/

[10] Using Scalable and Secure Web Technologies to Design a Global Digital Format Registry Prototype: Architecture, Implementation, and Testing, M. Geremew, S. Song, and J. JaJa, Proceedings of Archiving 2006, 92-95, May 23-26, 2006, Ottawa, Canada.

[11] Web Services Security (WS-Security) and Security Assertion Markup Language (SAML): http://www.oasis-open.org/specs/

[12] The Condor Project: http://www.cs.wisc.edu/condor/

7. Author Biography

Mike Smorul received his BS in computer science from the University of Maryland. He has a background in network and high performance computing system administration. More recently, he has worked as lead programmer for the UMIACS ADAPT project. Current project include developing a modular set of tools to aid in ingestion and long term digital stewardship of digital objects.

Joseph JaJa currently holds the position of Professor of Electrical and Computer Engineering with a joint appointment at the Institute for Advanced Computer Studies at the University of Maryland, College Park. Dr. JaJa received his Ph.D. degree in Applied Mathematics from Harvard University and has since published extensively in a number of areas including parallel and distributed computing, combinatorial optimization, algebraic complexity ,VLSI architectures, and data-intensive computing. His current research interests are in parallel algorithms, digital preservation, and scientific visualization of large scale data. Dr. JaJa has received numerous awards including the IEEE Fellow Award in 1996, the 1997 R&D Award for the development of software for tuning parallel programs, and the ACM Fellow Award in 2000. He served on several editorial boards, and is currently serving as a subject area editor for the Journal of Parallel and Distributed Computing and as an editor for the International Journal of Foundations of Computer Science