Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2011) 1–8
Procedia Computer Science

International Conference on Computer Science, ICCS 2011

Towards a Universal Viewer for Digital Content

Kenton McHenry, Michal Ondrejcek, Luigi Marini, Rob Kooper, Peter Bajcsy

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
{*kmchenry, mondrejc, lmarini, kooper, pbajcsy*}*@ncsa.illinois.edu*

Abstract

In this paper we present a distributed setup for the viewing of digital data within a large number of formats. Through the use of software servers 3^{rd} party software is automated and made available as functions that can be called within other programs. Using multiple machines containing software and running software servers a conversion service is built. Being built on top of 3^{rd} party software the conversion service is easily extensible and thus can be made to support a large number of conversions. We use this service to build a "universal" viewer by converting given files among a large set of file formats to a relatively small subset of formats that are renderable by a given viewer (e.g. web browser). We describe this service and the underlying software servers as well as the future directions we are planning on taking.

Keywords: conversion, preservation, software servers

1. Introduction

Scholarly articles today capture only a small cross section of the work they represent. Furthermore, the results of the work, the data obtained and software written, are in reality only available for a limited time period due to never ending changes in digital file formats and depended upon software. This inherent expiration date lowers the amount of information available for reproducing research and building on the top of it in the future. A possible way around this is for authors to publish everything, a complete reproduction of the scientific research done including all software used. This solution to this need for an "executable paper" could appear to be something akin to a digital file container that contains everything the authors used to produce their scientific results (something like a PDF or PDF/A document with extensions for software execution). We argue that this approach to digital content rendering is impractical and suggest an alternative approach.

Consider the problem of standardization that has faced the computer age since its birth. For something like an executable paper to exist we would have to agree on an accepted means of creating code, running that code, and storing information. The difficulties of an executable paper lie in the usual commercial reasons preventing standardization agreements, as well as in the large variation of specialized software/tools and formats used by a given scientific community. There is of course the virtual machine approach that would meet preservation needs by asking authors to embed a copy of their specific setup as a virtual machine published along with the paper. In this way one need only preserve the ability to execute the virtual machine in order to reproduce the obtained results. This approach is rather inelegant in that it is in essence a brute force approach saving everything potentially multiple times across different authors. In addition virtual machines tend to be large files making the potential for data sharing and long term storage an issue.

A different approach could be to address the problem of file format and software obsolescence by keeping the files current. The presented work addresses this through an extensible service that allows for a wide range of format conversions and through this the rendering of content from many formats. Our service-based approach is based on current trends that have been moving towards the idea of storing and processing digital data in a distributed manner in what is referred to as a "computer cloud". A consequence of this type of computing is that not all machines making up the cloud need to be equal. Many can be fairly lightweight machines and yet still perform beyond their capabilities by utilizing services provided by more specialized machines. It is this service-based approach that we take here. Rather than building a static all-encompassing file container that can store data and run software, we treat the file container itself as a temporally changing artifact in a cloud of distributed services that preserves the content. The concept of an "executable paper" is replaced in effect by a user customizable "view" of stored content which is created on the fly.

The following characteristics are essential to a system for scholarly work publication that allows for reproducibility in the long term: access to files any time from anywhere, interfaces for adding and recording metadata with files, automated conversion between file formats, automated execution of arbitrary software, and visualization capabilities. Towards addressing these requirements we have developed systems such as Medici [1] and Tupelo [2] to manage multimedia content with its metadata, CyberIntegrator [3] and Software Reuse Servers [4] to automate software execution and record execution trails, and Polyglot [5] to convert file formats and view their content.

2. File Format Conversion

The problems associated with an over abundance of file formats that store essentially the same content is something most are familiar with. While creating new file formats is often beneficial to software vendors in order to add attributes over time and lock in customers, it is a huge problem for users and archivists charged with preserving data. Just considering 3D data alone we have counted over 140 different file formats [6]. Having data distributed amongst so many formats makes it not only difficult to share data with others but also to preserve data in the long term. Consider proprietary formats where only the vendors responsible possess software that can fully load content from that format. Consider if they were to go out of business in a decade. Situations such as this where data exists on hardware but is locked away in no-longer supported formats occur all the time.

A solution both to data sharing across multiple users and data preservation is converting data to a popular standardized format that has an open specification. Such a format is much more likely to survive through time as anyone can re-create the loader from the specification if needed. Creating such a converter, however, is easier said than done. Implementing the needed loaders to store and extract content for the various formats is difficult due to the large number of file formats and the fact that many are closed specification proprietary formats. Rather than attempting this daunting task head on we take a round-about approach made possible by a piece of software called a *software server*.

2.1. Software Servers

Every programmer knows the value of code reuse. By reusing code you save time and avoid bugs. Clearly direct code reuse is only possible if one has access to the source code either directly or through a library. However, these are not always made available. As an alternative means of accessing and reusing code we look at the software produced from it. Software is built to be used by someone or something which in turn implies an interface of some sort. A software server attempts to re-impose an API on compiled code (i.e. software) by wrapping accessible functionality through whatever interface is provided in the software. We refer to this as *imposed code reuse* or *software reuse*.

Software servers exist as background processes on machines containing the software that it will utilize. These servers present a uniform means of accessing functionality within the software through one of various types of clients. Functionality within the software is controlled through wrapper scripts. In order to accommodate the various interfaces software can have software servers can be made to utilize any text based scripting language so long as it conforms to various naming and comment conventions [1]. As most modern software is built for human beings to use and in turn utilize graphical user interfaces, we have made extensive use of GUI based scripting languages such as AutoHotKey [2] and vision based scripting languages such as Sikuli [7]. Wrapper scripts are created for each desired operation within

[1] See scripting manual at http://isda.ncsa.illinois.edu/drupal/software/polyglot
[2] http://www.autohotkey.com

an application. With a set of such scripts over a handful of locally installed applications the software server is able to present tasks that it is capable of doing. These tasks can be treated as functions within an API. Users in turn can call these functions in true black box spirit, not worrying about the fact that what they are actually using is software.

Software servers support scripts for the operations: open, save, convert, modify, monitor, exit, and kill. The operations "open" and "save" deal with opening and saving of files within an application. The operation "convert" does both an open and a save within the same script. Since we are dealing with software rather than code these I/O operations are important not only for systems concerned with conversion (as will be described below) but also as a means of getting data in and out of the utilized application. The "modify" scripts are a general operation involving some functionality within the software that modifies the currently loaded data. An example would be a "blur" method within image software or "decimation" within 3D software. The remaining scripted operations are used to maintain the state of the software server. Software sometimes fails. The "monitor" scripts attempt to deal with frequently occurring exceptions to the norm that may be encountered when interacting with an application (e.g. dialogues for overwriting files or for updating the software). The scripts for "exit" and "kill" deal with exceptions that can not be dealt with and forcibly exit the software allowing the software server to try again.

2.1.1. Java API

A Java API to the functionality captured by a software server is made available through the SoftwareReuseClient and Task classes. An example of how these are used is shown here for the purpose of converting a file to a different format:

```
Task task = new Task(new SoftwareResueClient(server, port));
task.add(Blender, convert, ./heart.wrl, heart.stl);
task.add(A3DReviewer, open, heart.stl, );
task.add(A3DReviewer, export, , heart.stp);
task.execute(.);
```

Each task is made of subtasks containing a software, an operation, an input (possibly null), and an output (possibly null). Though subtasks can be issued directly via the SoftwareReuseClient class this is discouraged as operations such as "open" and "save" are not "thread safe". In other words if another request comes in to use the same software between an "open" and "save" request the two requests would interfere with one another. The Task class ensures operations within subtasks are carried out all at once.

2.1.2. Restful Interface

The software servers can also be run to expose a restful interface [3], allowing pretty much any programming language, scripting language, or even browser access to captured functionality via URLs. Software reuse servers utilizing the restful interface can also be setup to connect to other software servers so as to provide a greater set of functionality, enhanced robustness, and parallelism. An example illustrating a conversion via the restful interface would be posting a file to the given URL:

```
http://host:port/software/Blender/convert/stl/
```

where *.stl will be the desired output format. In general software reuse endpoint URLs take the form of:

```
http://host:port/software/<Software>/<Task>/<Ouput Format>/<Input File>
```

Here tasks are automatically generated with the necessary input/output operations.

2.2. NCSA Polyglot

Most software supports a small handful of formats to allow for some level of data sharing. We document such software along with the input and output formats they support. Next, we use this information to create software server wrapper scripts for "open" and "save" operations and setup one or more software servers. Software can be run

[3] http://www.restlet.org

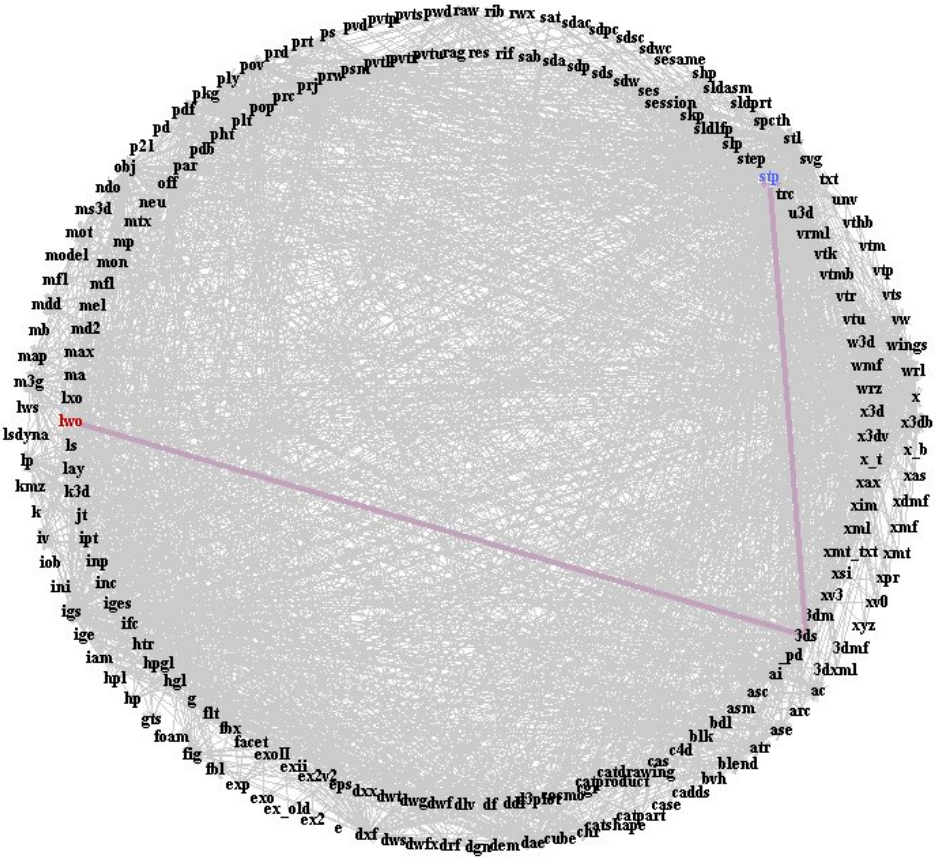

Figure 1: *A visualization of the input/output graphs used by a Polyglot service. The vertices within the graph represent file formats. The edges are directed and represent applications running on a remote software server that are capable of converting from the source format to the target format. A conversion path between file formats is found by searching for a shortest path between a source and target vertex. If the graph is un-weighted this path will be the one involving the fewest number of applications. If the graph is weighted by an estimated information loss on each application/edge, then this path will be the one with the least overall information loss.*

across various machines and need not be installed on every one. Software servers are combined through a Polyglot [4] server which examines all the input/output operations available on the software servers and uses this information to construct an input/output graph (see Figure 1) containing available conversion paths between the union of utilized file formats. Note, this is different from the service provided by the combined software server restlets. The restful interface provides all tasks, not just those associated with format conversion, and does not consider chains in order to complete a conversion. Through the construction of the I/O-graph the polyglot service is capable of performing conversions between a larger set of formats by utilizing more than one software to get from a source format to a target format. Where as software servers are focused on providing access to software functionality, the Polyglot server is focused on converting between an input and an output format.

2.2.1. Java API

Like the software servers Polyglot can be used by an API provided by a PolyglotClient class:

```
PolyglotClient polyglot = new PolyglotClient(host, port);
polyglot.convert("C:/Users/Foo/heart.wrl", "C:/Users/Foo", "stp");
```

[4] **n.** one who speaks many languages

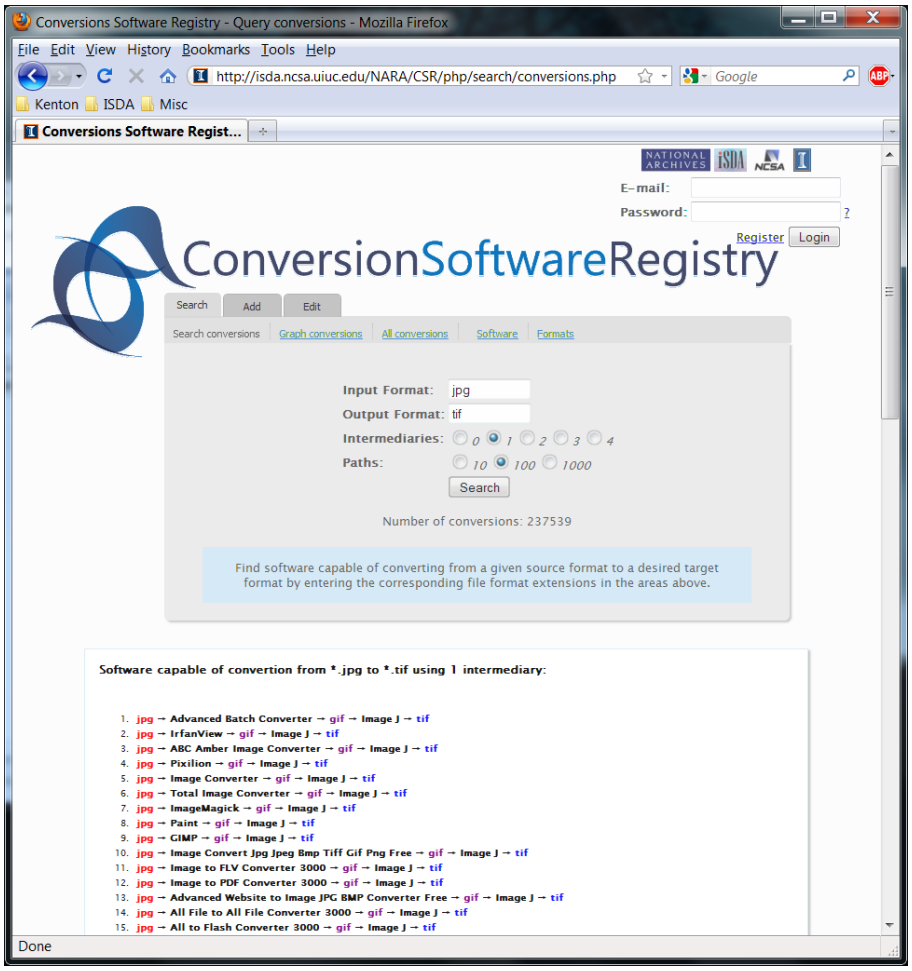

Figure 2: *The web interface to the Conversion Software Registry. Users can query the database by entering the extensions of an input and output file format to find applications that are capable of performing the conversion.*

where the first parameter is an input file, the second an output path, and the third the extension of the desired output format.

2.3. The Conversion Software Registry

Though by no means a huge chore the most time consuming part of setting a polyglot server and its underlying software servers is writing the wrapper scripts for the utilized software. To facilitate this process we have created the Conversion Software Registry (CSR) [8]. The CSR [5], a complement to file format registries like PRONOM [6] and GDFR [7], is an on-line database that emphasizes software and the inputs/outputs they support. The database is currently populated with conversion information on 2,052 applications. Users can search the registry by entering an input format extension and a desired output format extension in order to view all software, or chains of software, capable of making the conversion (see Figure 2). The CSR web site also allows users to directly search a graphical representation of the I/O-graph by selecting a subset of software, selecting source/target formats and viewing conversions paths. With this information one can identify suitable software to use for their particularly needs (i.e. the conversions that are required to be performed).

[5] http://isda.ncsa.illinois.edu/NARA/CSR
[6] http://www.nationalarchives.gov.uk/aboutapps/pronom
[7] http://www.gdfr.info

As a secondary service we have begun using the CSR as a repository for software wrapper scripts. To take advantage of this we created a ScriptInstaller tool that is included with the Polyglot installation. When executed on a newly created software server the tool will scan the local system for installed software (for example using the registry on Windows machines to find software) and then query the CSR. If scripts are found for the installed software they are downloaded and configured to run on the local system. In addition to these scripts the CSR has also become a repository for test data. The ScriptInstaller can be made to use this data as input files for the installed software and automatically attempt a specified number of conversions in order to determine if the system is setup correctly to use the software. With the CSR acting as script/data repository creating new software servers and adding their conversion capabilities to a Polyglot service can be a fairly simple and automated process.

2.4. Versus Comparison API

Polyglot was designed through a need by the U.S. National Archives and Records Administration to measure information loss across file format conversions [5]. To do this a library of content comparisons called Versus is being developed. Versus imposes a uniform API on comparison measures for data stored in a number of representations. With a small number of implemented file loaders files can be loaded into one of the supported representations and compared. By comparing files before and after a conversion Polyglot is able to make a measure specific estimate of the amount of content loss incurred. These information loss estimates, which usually take the form of a distance between two files, are normalized and averaged over a large number of conversions and then used as weights on the edges of the I/O-graph. With these weights one can search for conversion paths between formats that result in the least amount of estimated information loss. One can also ask questions such as which software performed conversions with the least information loss or which format resulted in the least amount of information loss from other formats (i.e. which is the best format for preservation purposes).

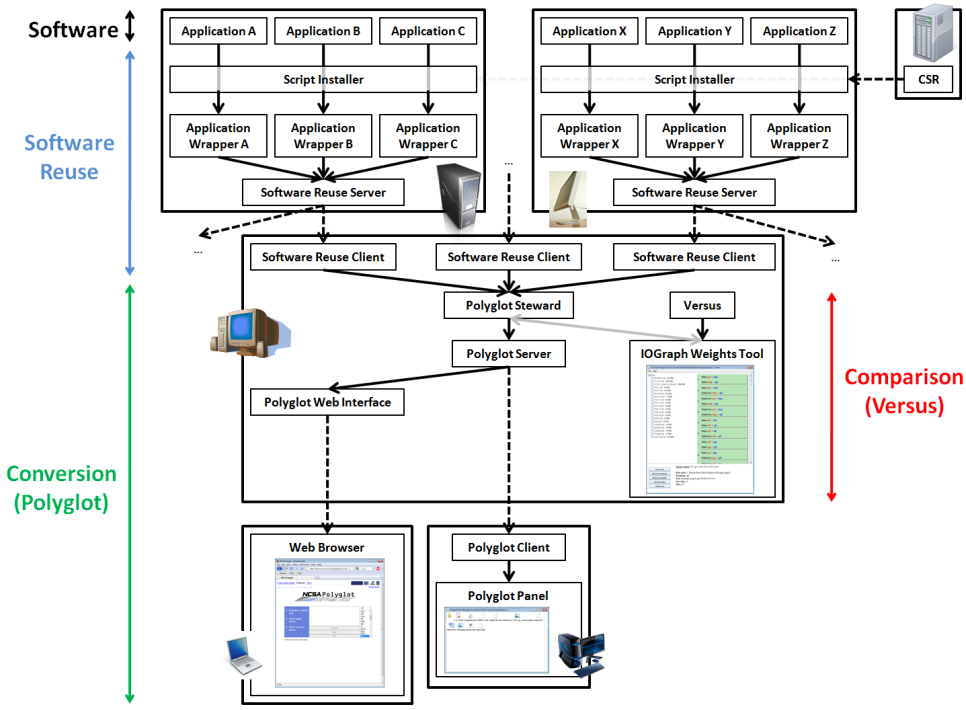

Figure 3: *An overview of the Polyglot service. At the lowest level the system is built upon a number of "worker bee" nodes that run software servers for scripted locally installed software. A Polyglot server combines the input/output functionality among these software servers within an I/O-graph which it uses to search for and execute multi-step conversions between the union of supported file formats. Various clients can take advantage of this service from command line tools, to file managers, to web browsers.*

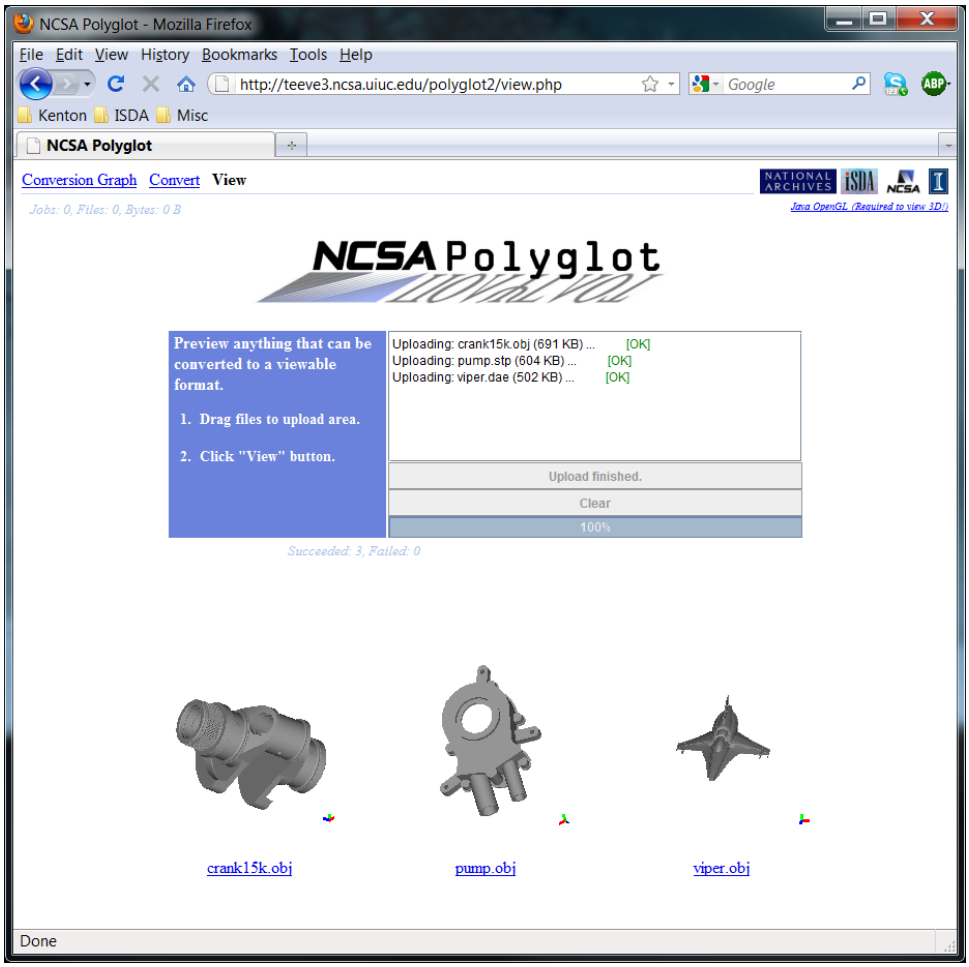

Figure 4: *The web based "universal" viewer included as part of the Polyglot web interface. Users can drag and drop files from their local machine to the area at the top of the page. When the user then presses the "View" button the files are submitted to the Polyglot service for conversion to one of the pre-selected formats that are deemed viewable by most browsers. In the example shown here three different 3D files of three different formats are uploaded. The included 3D viewer applet supports the "*.obj" format thus all other formats are converted to this format. When the conversion is complete a viewer instance is started for each file allowing the user to view and interact (rotate, translate, scale) the 3D models.*

3. "Universal" Viewing

Figure 3 outlines the Polyglot system and how it interacts with the component services described above. Because Polyglot servers are built on top of one or more software servers it is only capable of performing conversions that are achievable from the union of the underlying software. However, the service is easily extendible by adding additional software servers with additional software. Due to this extensibility a Polyglot service could in theory be made to be a "universal converter" between any pair of formats assuming a suitable set of software was obtainable.

With such a conversion service available we have explored the idea of a web based "universal" viewer. Web browsers support the viewing of a variety of file types. With plugins this can be extended if the needed plugins are available. However, with a "universal" converter we could view (or preview) the content of potentially any file. By choosing a format that the browser supports and converting files in other formats to this format automatically behind the scenes we can give the illusion of a viewer that supports a very large number of formats.

We have implemented a proof of concept of this viewer as part of Polyglot [8]. By clicking in the view pane of the Polyglot main page one is presented with a page containing an area where files can be dragged and dropped to.

[8] http://polyglot1.ncsa.illinois.edu

Once files have been dropped here and the "View" button pressed the files are submitted to a Polyglot service for conversion into a pre-selected list of formats that are supported by most browsers (e.g. "*.jpg" for images, "*.txt" for documents, "*.obj" for 3D models utilizing an included Java applet viewer). If a conversion path exists from the given format to one of the viewable formats through the software servers installed 3^{rd} party applications then the file format conversion is automatically carried out. When completed the resulting file is retrieved and displayed in the browser. In Figure 4 we see an example where 3D files of various formats are being viewed through this tool.

4. Conclusion

The described file format conversion and content rendering service, utilizing software servers populated with relevant software, has potential as a part of a dynamic scholarly paper built on an author's code and data. Regardless of the format used by the authors their data can be previewed. By using this distributed setup the computer where the data is viewed need not have any of the needed software. By storing the data in distributed file systems such as those mentioned earlier we can take on archival from the onset. In addition, the utilized software reuse servers have potential far beyond the conversions they are used for here in the Polyglot service. We envision them as a means of sharing and using platform specific code within a cloud paradigm in a manner somewhat similar to that of [9, 10] but without the software interface restrictions. We conclude by stating that the presented "universal" viewer, in combination with the other mentioned distributed services, has the potential to meet the requirements of an "executable paper" allowing scholarly work to be more permanent, more accessible, and better reproducible. In the future we plan on integrating NCSA Polyglot into Medici [9] [1], an open source web based content repository allowing for the sharing of data on a customizable privately owned and controlled system. Medici allows for the uploading of arbitrary files and presents previews for those that it recognizes. By utilizing the described conversion service we would greatly expand its previewing capabilities by allowing it to convert unrecognized files to formats that are recognized.

Acknowledgement

This research has been funded through the National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreement NSF OCI 05-04064 by the National Archives and Records Administration (NARA).

References

[1] L. Marini, R. Kooper, J. Futrelle, J. Plutchak, A. Craig, T. McLaren, J. Myers, Medici: A scalable multimedia environment for research, The Microsoft e-Science Workshop.
[2] J. Futrelle, J. Gaynor, J. Plutchak, J. Myers, R. McGrath, Tupleo: a framework for e-science knowledge spaces, The Microsoft e-Science Workshop.
[3] R. Kooper, L. Marini, B. Minsker, J. Myers, P. Bajcsy, Cyberintegrator: A highly interactive scientific process management environment to support earth observations, Geoinformatics Conference.
[4] K. McHenry, R. Kooper, L. Marini, P. Bajcsy, Designing a scalable cross platform imposed code reuse framework, The Microsoft e-Science Workshop.
[5] K. McHenry, R. Kooper, P. Bajcsy, Towards a universal, quantifiable, and scalable file format converter, The IEEE Conference on e-Science.
[6] K. McHenry, P. Bajcsy, An overview of 3d data content, file formats and viewers, Technical Report ISDA08-002.
[7] T. Yeh, T. Chang, R. Miller, Sikuli: Using gui screenshots for search and automation, UIST.
[8] M. Ondrejcek, K. McHenry, P. Bajcsy, The conversion software registry, The Microsoft e-Science Workshop.
[9] C. Pancerella, The use of agents and objects to integrate virtual enterprises, SANDIA Report 8226.
[10] R. Whiteside, E. Friedman-Hill, R. Detry, Pre: A framework for enterprise integeration, SANDIA Report 8505C.

[9] http://medici.ncsa.illinois.edu