CONAT20046005

CAD – WEBDAV ADAPTOR: PREMISE FOR A LIGHT PLM SOLUTION

[1]Covaciu, Dinu[*], [2]Brădău, Horia, [1]Preda, Ion

[1]Transilvania University Braşov, Romania, [2]kPrism ES, Woodbridge, Canada

KEYWORDS

CAD, PLM, collaborative engineering, WebDAV, DeltaV, DASL

ABSTRACT

In today's dynamic engineering environment, including the automotive area, collaboration is a hard requirement. More than in other disciplines, the CAD design process can't be conceived outside a team. Central (secure) repository, remote access, metadata, access control and versioning are mandatory tools for a design team in order to be effective and successful. PDM/PLM platforms offer solid solutions for big companies, which use expensive CAD platforms, but they are a luxury that smaller companies/teams can't currently afford.

The general need for collaboration lead the Internet community and groups (W3C, IETF) to define WebDAV (Web Distributed Authoring and Versioning), a HTTP based protocol that allows teams to collectively work on the same document.

The paper presents a WebDAV adaptor that enables CAD clients to access a remote WebDAV server. It features access control, metadata management, locking (checkin/checkout), versioning, part relations (in assemblies), and enables teams for distributed design process.

The adaptor was implemented for Unigraphics, as a C/C++/OpenUg application and it can be used with any server implementing the WebDAV specifications.

Designed for CAD activities, the proposed solution can be easily adapted for other domains.

MAIN SECTION

PLM (Product Lifecycle Management) seeks(1) to provide horizontal collaboration throughout a product's lifecycle, from concept to disposal. PLM is not just product data management (PDM); PLM strives to offer authoring, managing and sharing tools for data that is created and used throughout a product's lifecycle by all that touch or have a need for information on a product.

To date, the two biggest manufacturing market segments that have embraced PLM are automotive and aerospace. These two industries were among the first to implement and integrate PLM into their businesses just based on the sheer complexity of the products they manufacture and on the multitude of suppliers they have to deal.

A PLM system must have at least most of the following subsystems or modules:

- CAD/CAM;
- Bill of materials (BOM) processor;
- Configuration management;
- Computer-aided process planning (CAPP);
- Database management;
- Data communications;
- Visualization;
- User interface;
- Application programming interfaces (API)

Ideally, the PLM system should be tied to supply chain management (SCM) and enterprise resource planning (ERP) systems, and herein lies the fact that PLM can be tricky to implement. The biggest challenge in successfully deploying a PLM system on any scale is understanding and managing the full range of data types that can be generated within and throughout a PLM system.

Most of the PLM subsystems are tied directly to manufacturing. In manufacturing, PLM systems must be customized and fine tuned according to unique business models in terms of which specific PLM components are included.

PLM technology optimises and exploits the interaction of information with different people in a business. Simply put, this is called workflow, a complex chain of processes that transforms thoughts into actions and, ultimately, into tangible products. In the end, PLM is as much about business strategy as it is about a product's lifecycle.

WEBDAV

WebDAV stands (2) for “Web Distributed Authoring and Versioning” and represents an extension to the HTTP/1.1 protocol that allows clients to perform remote web content authoring operations.

WebDAV provides a coherent set of methods, headers, request entity body formats, and response entity body formats that provide operations for: **properties** (the ability to create, remove, and query information about documents, such as their authors, creation dates, etc. and also the ability to link documents of any media type to related documents), **collections** (the ability to create sets of documents and to retrieve a hierarchical membership listing), **locking** (the ability to keep more than one person from working on a document at the same time; this prevents the “lost update problem”, in which modifications are lost when a second author writes changes without merging the first author's changes), and **namespace operations** (the ability to instruct the server to copy and move Web resources).

Within the HTTP/DAV/DeltaV family of specifications, a document on a Web server is known as a *resource*. Like objects in object-oriented languages, resources have a state and operations on that state. The state of a WebDAV resource comes in two parts: a *body* (that contains the primary content, like the CAD model or drawing) and *properties*. The operations on resources are termed *methods*.

Collections are resources that contain a set of URIs (Uniform Resource Identifier), which identify member resources (like a directory in a file system).

Properties are pieces of data that describe the state of a resource (they are data about data). A property is a name-value pair that contains descriptive information about a resource. There are two

categories of properties: “live” and “dead”. A **live property** is a property whose value is both computed and controlled by the server. A **dead property** is a property whose value is controlled by the client, and stored by the server.

HTTP protocol operations are called **methods**, and WebDAV adds seven new methods to the set of methods defined by HTTP/1.1 (GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE). The WebDAV methods provide overwrite protection (LOCK, UNLOCK), metadata management (PROPFIND, PROPPATCH), and namespace management (COPY, MOVE, MKCOL). Generally, a user of a WebDAV-enabled authoring tool is unaware of the WebDAV protocol use. The WebDAV protocol is designed to be integrated into existing authoring tools, adding Web-based remote authoring capabilities to the tools already familiar for the users. To date, this has been a successful strategy, with WebDAV support in document authoring tools such as Microsoft Word, PowerPoint and Excel (via the “Web Folders” feature).

In HTTP/1.1, method parameter information was exclusively encoded in HTTP headers. Unlike HTTP/1.1, WebDAV encodes method parameter information either in an Extensible Markup Language (**XML**) request entity body, or in an HTTP header. The use of XML to encode method parameters was motivated by the ability to add extra XML elements to existing structures, providing extensibility. In addition to encoding method parameters, XML is used in WebDAV to encode the responses from methods, providing the extensibility advantages of XML for method output, as well as input.

DELTAV

The WebDAV Working Group, when it went to create document management features found that versioning was critical and included it from the start. As WebDAV progressed it was found that versioning was very hard and that it required a special attention. The DeltaV protocol is an extension to the WebDAV protocol. DeltaV (3) is picking up where WebDAV left off, extending the protocol with versioning and configuration management support.

Versioning is the ability for a resource to be checked into a version controlled system where it has multiple revisions that are tracked and can have multiple successor and predecessor relationships. The server will maintain those relationships, report the revision history, and control the write able access to these revisions using check in/out operations. Parallel development provides more resource availability in a multi-user environment. So, multiple users can checkout the same revisions of a resource, then check that they has the same revision and finally merge them back later on as appropriate. Configuration management means to bring together consistent revisions of resources.

To the base provided by HTTP and WebDAV, DeltaV adds 11 additional methods. Versioning capability is provided by the methods VERSION-CONTROL, CHECKIN, CHECKOUT, UNCHECKOUT, and REPORT. An unversioned resource is put under version control with VERSION-CONTROL. While under version control, a typical editing process begins with CHECKOUT, involves one or more writes (PUTs) to the resource, and ends with a CHECKIN. An editing session can be aborted using UNCHECKOUT. The version history of a resource can be retrieved using REPORT. Unique user-friendly names can be associated with specific versions using LABEL. The default visible revision can be set using UPDATE. The others methods added by DeltaV are MERGE, MKACTIVITY, MKWORKSPACE and BASELINE-CONTROL.

DASL

DASL represents WebDAV SEARCH, an application of HTTP/1.1 forming a lightweight search protocol to transport queries and result sets and allows clients to make use of server-side search facilities.

The basic usage of DASL follows these steps:

• the client constructs a query using the DAV:basicsearch grammar;
• the client invokes the SEARCH method on a resource that will perform the search (the search arbiter) and includes a text/xml or application/xml request entity that contains the query;
• the search arbiter performs the query;
• the search arbiter sends the results of the query back to the client in the response; the server must send an entity that matches the PROPFIND response.

Implementation of a WebDAV client for Unigraphics

Most of the CAD systems offer a C++ based API: CAA for CATIA V5, Pro/Toolkit for Pro/Engineer, OpenUG for Unigraphics, Ideas-Open for Ideas, ObjectARX for AutoCAD. A WebDAV client writen in C++ will be suitable for all these CAD systems, with different interfaces. For this reason, we developed a C++ wrapper over **Neon**, an existing WebDAV client library, written in the C programming language.

The **Neon** library, developed by Joe Orton (5), features:

- high-level interface to HTTP and WebDAV methods (PUT, GET, HEAD, etc.);
- low-level interface to HTTP request handling, to allow implementing new methods easily;
- persistent connections;
- basic and digest authentication;
- SSL/TLS support using OpenSSL (including client certificate support);
- Generic WebDAV „207" XML response handling mechanism;
- XML parsing using the Expat or Libxml parsers;
- easy generation of error messages from „207" error responses;
- WebDAV resource manipulation: MOVE, COPY, DELETE, MKCOL;
- WebDAV metadata support: set and remove properties, query any set of properties (PROPPATCH/PROPFIND).

Neon library don't offer support for DeltaV and DASL (versioning and searching), so we have to extend the library with new functions. The extended features we added to the library consist mainly in support for REPORT and SEARCH methods.

The authors written an original C++ wrapper over Neon, named **Neonplus**. It defines a main object called WebdavDlg. The objects defined in the *Neonplus* library can be called in custom applications developed for different CAD platforms which support C++, like Unigraphics, Inventor, Pro/Engineer, Catia, Ideas, or AutoCAD.

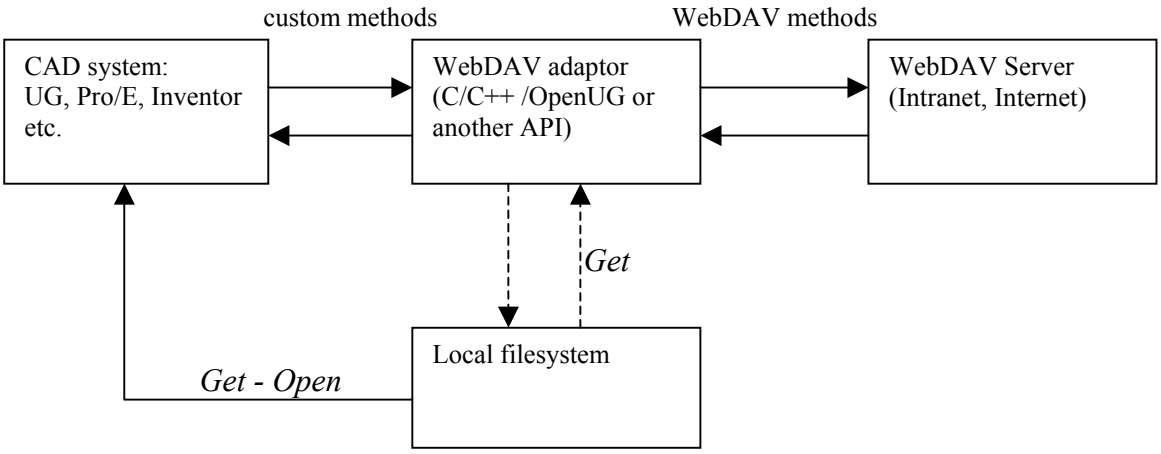
An example of using *Neonplus* for Unigraphics is below:

```
/* MKCOL DIALOG -----------------------------------------*/
int MKCOL_ok_put ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );
       char *coll = "";
       WebdavDlg *dlg;
       dlg = new WebdavDlg(user_name, user_pwd, server_path);
        //create a new collection: (name given by dialog)
        callback_data->item_attr = 2;
        callback_data->item_id = MKCOL_WIDE_S_0;

        UF_STYLER_ask_value(dialog_id,callback_data);
        coll = callback_data->value.string;
        log->write("\nCount: %d",callback_data->count);
        log->write("\nCollection: %s",coll);
        dlg->Mkcol(coll);
             delete dlg;
        free(coll);
     UF_terminate ();
    return (UF_UI_CB_EXIT_DIALOG);
}  // Mkcol
```

The black lines in the code above are the only used to call the Webdav functions from the *Neonplus* library. These lines are called in the same way also under other CAD platforms programming interface.

A diagram that explain how the custom defined methods and the WebDAV methods works in a CAD system is shown below:



The same diagram (in the above example *Get*) applies for the other methods (*Put, Lock/Unlock, Mkcol, Report, Search*).

The WebDAV server used for our demo was Jakarta Slide.

Table 1 - Examples of implemented methods

Client Side	*Server Side*
Get Part – ask for the last (current) version of the part or assembly (file)	- identify the last version - check if the file is unlocked - lock the file - send file to client
Put Part - save a modified part as current version	- save the received file as a new version - unlock the file for the other users
Get/Put Properties – add custom data to the part (according to the internal standards)	- to add/change a property, locks the file before and unlock them after
Get previous version – ask for a previous version of a part (assembly) and get the corresponding file from server	- generate a report (a list of all saved versions for the requested file) - find the selected version in the *history* collection - send the file to the client, like for the *Get Part* request
Search for parts – search for parts on server, based on metadata (properties)	- execute the *Search* method - generate a list with files (parts) that matches the request

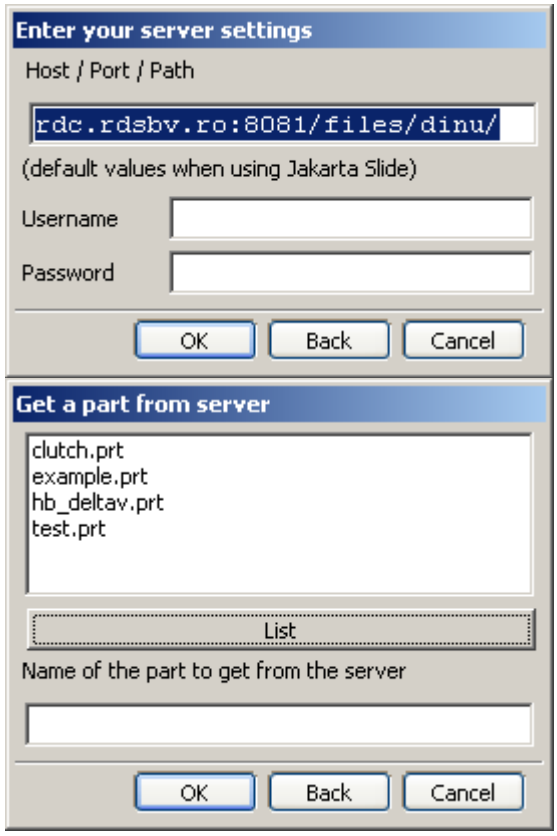
Example of using the test application to *get/open* a part from the server

First, start Unigraphics. Using the **Neon_test** menu, log on to the server.

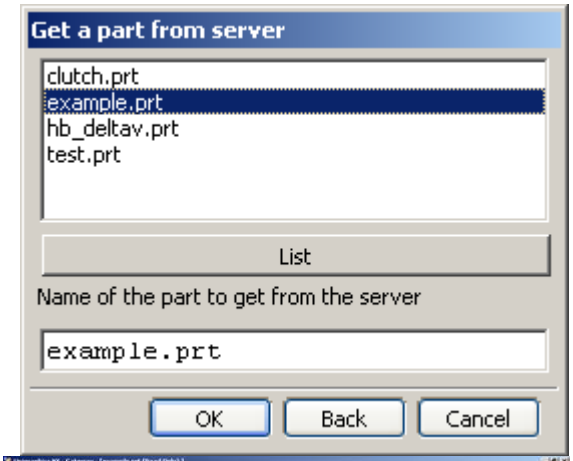
We suppose the path already exists on the server (in this example, /files/dinu/). The path must be terminated with a / (slash).

The user must enter his username and password to give access to that path on the server.

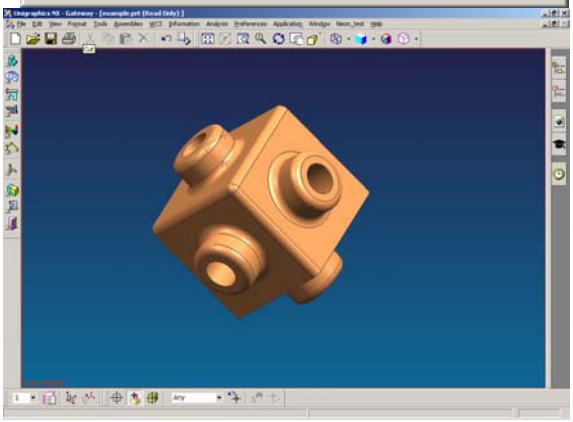
From the *Neon_test* menu, select **Get**. The following dialog box will be displayed:



The list contains the files with extension *.prt* on the current path. Select an item of this list and then **OK**.



The selected part will be opened:



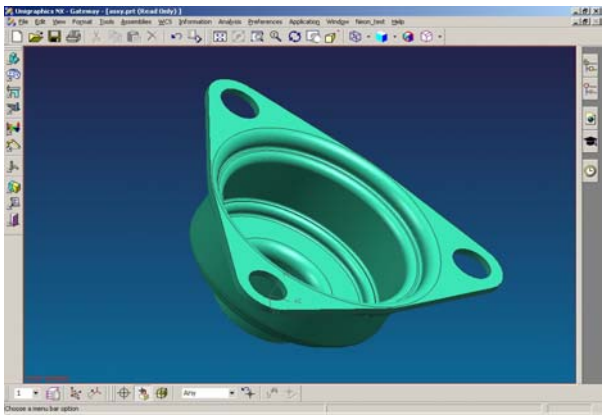
To *put* a part on the server, first open that part in Unigraphics, that select **Put** from the *Neon_test* menu.

The other regular WebDAV methods are implemented in a similar way. A special attention must give to the assemblies.

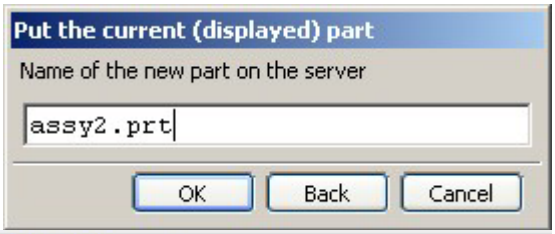
The methods **Get** and **Put** works apparently like for any other part. When put an assembly, a new property, named *component_count*, is set. The value is equal with the number of the parts in the assembly. For each component part, is set another property, named *componentX*, where *X* is an integer, representing the index of the component. The value of these properties is the name of the component part. When getting a part, the program checks if it is an assembly by reading the property *component_count*. If this property is defined and the value is greater than 0, the part is an assembly and the properties *componentX* will be read. Each component will be also copied from server using the same method: *Get*.

Example of using the test application to *put* an assembly to the server

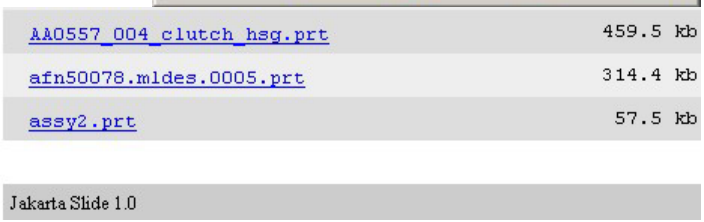
Suppose the attached assemly is opened in Unigraphics.



To put the assembly on server, select from the menu **Put**.
Write a name for the assembly and press OK.



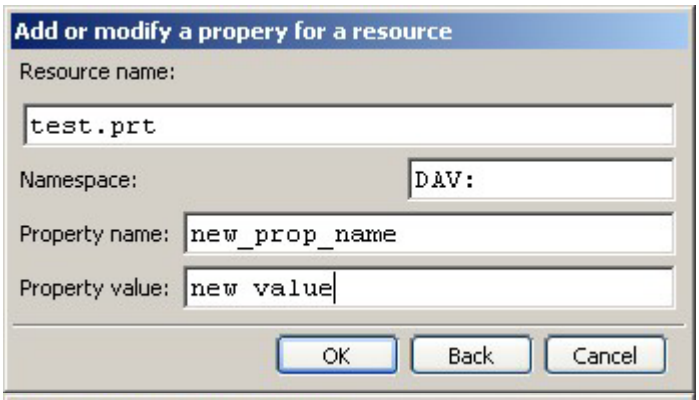
Check the result on server. There are three new items: the given assembly and two component parts.



To add custom information (properties) to a part, we use the method **Proppatch**. The additional information attached to a part can be retrieved using the method **Propfind**.

Example of using the test application to add/retrieve custom data

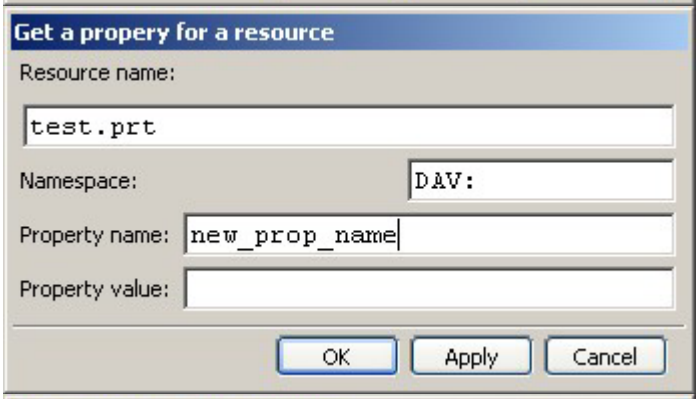
PROPPATCH: set a (new) property for a resource



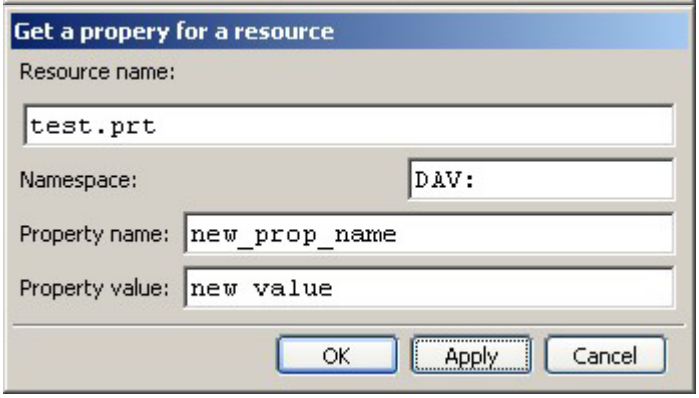
PROPFIND: fetch a property from a resource

Enter the resource (part) name – it is not necessary to have the part opened on the local system.

Enter the property name and click **Apply.**

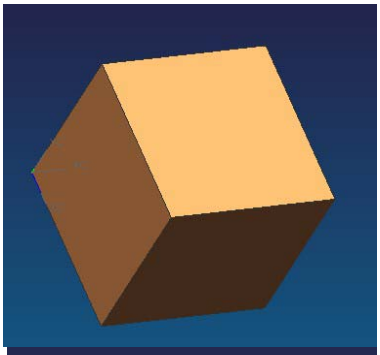


The result:



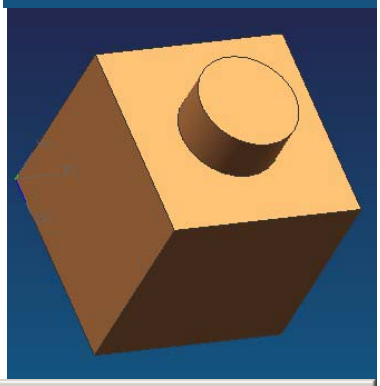
The most challenging part of the application is to add versioning capabilities to Unigraphics. Next we present an **example of using the test application to save a new version of a resource (part) on the server and retrieve a previously saved version.**

Open a new part, with the name "example".
Create a very simple solid.



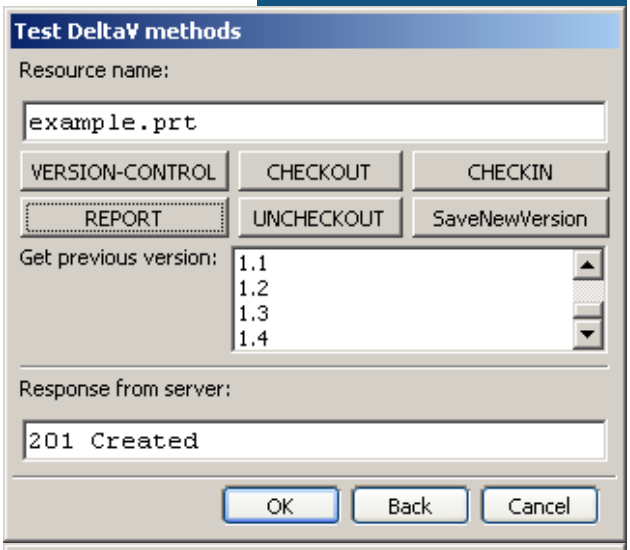
From the *Neon_test* menu, select **Put**.
Enter the name "example.prt".
This is a simple PUT, and will create the initial revision on server.

Modify the part.



From the *Neon_test* menu, select **DeltaV_test.**

Enter "example.prt" as the resource name, and press **SaveNewVersion**.
The result must be "201 Created".



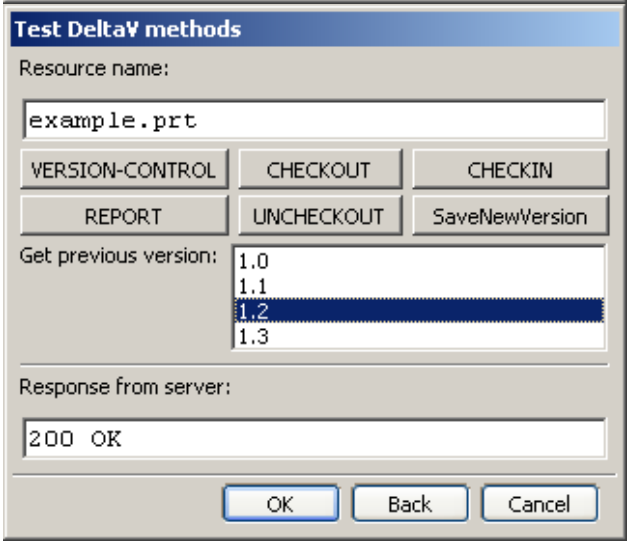
You can continue to add new features and save new versions on the server.

In the same dialog box (*DeltaV_test*), press **REPORT**.

The content of the list containing the previous versions is changed. The new list contains all the available versions for our part.

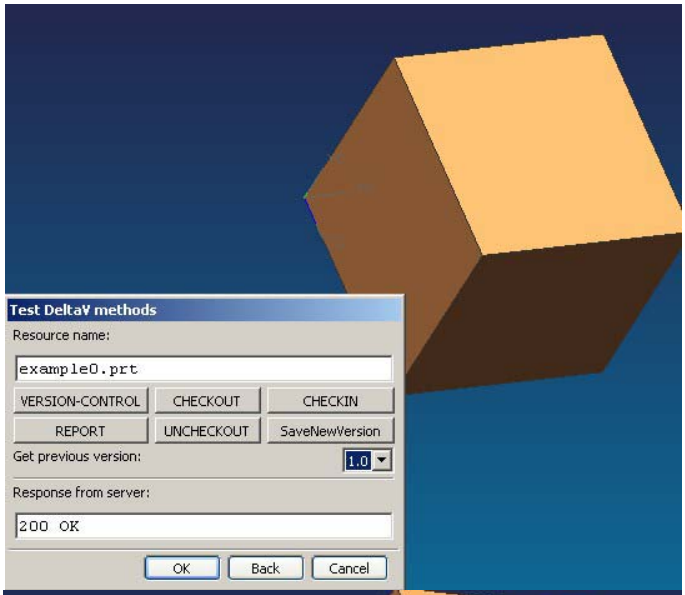
To get a previous version from server, click in the list on the desired version name.

The response from server must be "200 OK".



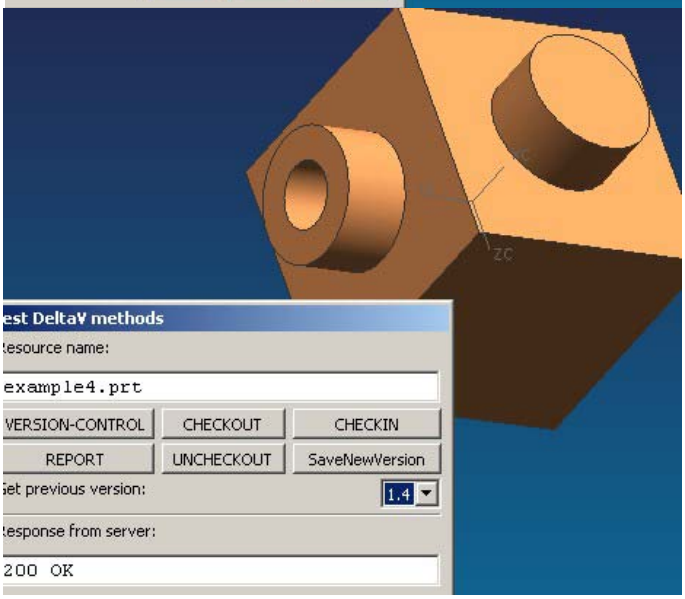
Now the *resource name* must be the name of the part on the local system. If the filename already exists on your hard disk, the program will append an underscore "_" in front of the existing name. The *resource name* can remain the same. All the files are saved in the root of your disk (c:\).
The result from server must be "200 OK".

Here is the version 1.0.



In the same way, retrieve another version.

Here is the version 1.4.



The **search** (DASL module) is done using metadata (properties).

Select from the pull-down menu **Search.**

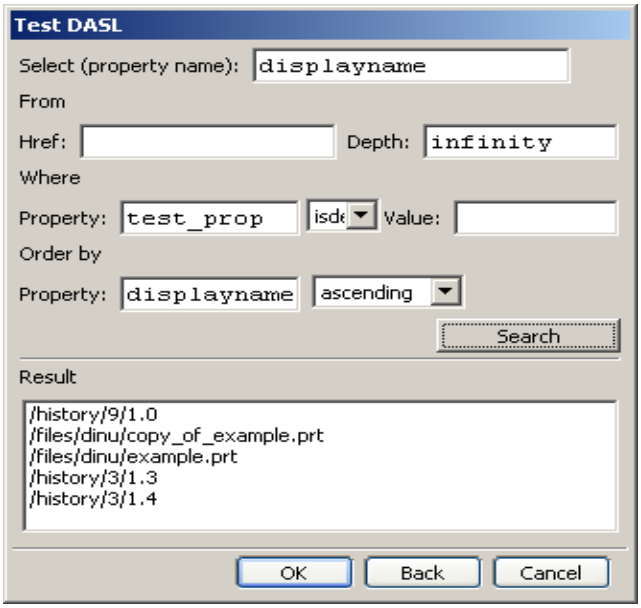
Some fields will be filled-in by default.

A search example is attached.

In the search result, the lines */history/3/1.x* are history versions of the file *example.prt*.

This example can be written like that:

```
SELECT property "displayname" FROM
current path, with depth: infinity,
WHERE property "test_prop" is
defined, ORDER by property
"displayname", ascending
```



Of course, in this paper are presented only few of the capabilities of the application. Our goal was to demonstrate how the *Neonplus* library could be used to implement WebDAV under a CAD system. The custom data and version information could be used to connect the CAD system to the enterprise PLM system.

CONCLUSIONS

For many years, CAD was predominantly a tool for individual creativity. Now the enterprise managers are focused to use the creative capital of the whole team. Designers will need to work at a higher conceptual level in design terms, but within constraint systems that have been predefined. This is collaborative engineering.

The aspects presented above represents an original contribution of the authors to this domain. This demo application can be the basis for a light PLM solution. There is still much to do until this solution will be used in real design and data management activities, but we hope that our future work will allow us to put the entire PLM system on the market.

REFERENCES

(1) Jeffrey Rowe, "Is PLM software the real deal?", Advanced Manufacturing magazine, http://www.advancedmanufacturing.com, May/June 2004
(2) IETF, "RFC2518: HTTP Extensions for Distributed Authoring – WEBDAV", http://www.webdav.org, February 1999
(3) Jim Whitehead, "DeltaV: Adding Versioning to the Web", http://www.webdav.org/deltav/WWW10/deltav-intro.htm, 2001
(4) James J. Hunt, Jurgen Reuter, "Using the Web for Document Versioning: An Implementation Report for DeltaV", http://www.ipd.uka.de/~reuter/publications/deltav.pdf , 2000
(5) Joe Orton, The Neon library, http://www.webdav.org/neon, 2000-2004