Integration of Version Control in Digital Content Creation

Christopher Bergqvist

Luleå University of Technology
BSc Programmes in Engineering
Skellefteå Campus
Abstract

In the production of computer graphics, minimizing risk and maximizing efficiency is becoming more and more important. One tool that can be used by the production teams to this affect are version control systems.

This thesis describes the work of producing a prototype for the plugin for one of the most used applications for digital content creation, Alias Maya. The plugin (Maya Asset Manager) integrates version control into the file management process in order to ease the use of it and make computer graphics artists more productive when working in a team.

The plugin prototype uses the CVS console client as a backend and C++ and MEL to interact with the main application's interface.
Sammanfattning

Inom produktionen av datorgrafik blir minimering av risk och maximering av effektivitet allt viktigare. Ett verktyg som kan användas för att uppnå en del av detta mål är versionshanteringssystem.

Denna rapport beskriver arbetet med att producera en prototyp för ett s.k. plugin för ett av de mest använda programmen för skapande av datorgrafik, Alias Maya. Pluginet (Maya Asset Manager) integrerar versionshantering i filhanteringsprocessen för att underlätta dess användning och göra datorgrafiker mer produktiva när de arbetar tillsammans med andra i en grupp.

Pluginprototypen använder konsolklienten för CVS för att integrera sig själv med versionhanteringssystemet och C++ och MEL för att integreras med huvudprogrammets användargränssnitt.
Integration of Version Control in Digital Content Creation

Acknowledgements

I would like to thank:

• Håkan Wallin at Data Ductus for the original idea for a Maya Asset Manager.
• Andreas Wiklund, my mentor at Data Ductus, for writing a paper on the idea and for being patient with my initial insecurity regarding Maya development.
• Computer Graphics student Oskar Holmstrand, for providing valuable feedback and an out-of-the-box perspective.
Contents

1 Introduction ............................................................................................................. 1
  1.1 Problem statement ................................................................................................. 1
  1.2 Background to the problem ................................................................................... 1
    1.2.1 Performing DCC with Maya ........................................................................... 1
    1.2.2 Version control ............................................................................................... 1
  1.3 Requirements of the prototype .............................................................................. 1
  1.4 Stylistic Remarks ................................................................................................... 2
  1.5 Word list ............................................................................................................... 2

2 Methods .................................................................................................................... 4
  2.1 Tools ..................................................................................................................... 4
    2.1.1 MEL ................................................................................................................ 4
    2.1.2 Maya C++ API ............................................................................................... 4
    2.1.3 CVS ................................................................................................................ 4
  2.2 Development process ............................................................................................ 4
    2.2.1 Research ........................................................................................................ 4
    2.2.2 Customer/user discussions ............................................................................ 5
    2.2.3 Design ............................................................................................................ 5
    2.2.4 Implementation ............................................................................................... 5
    2.2.5 Testing ............................................................................................................ 5

3 Results .................................................................................................................... 6
  3.1 Installation ............................................................................................................. 6
  3.2 Replacing the calling of original dialogues and windows .................................... 6
    3.2.1 File menu ....................................................................................................... 6
    3.2.2 Main tool bar ................................................................................................ 6
    3.2.3 Hotkeys .......................................................................................................... 7
    3.2.4 Unsaved changes ........................................................................................... 7
  3.3 Browser window .................................................................................................. 7
    3.3.1 Open Scene Window .................................................................................... 8
    3.3.2 Save Scene As Window ................................................................................. 9
  3.4 History Window .................................................................................................. 9
  3.5 Preferences Window ............................................................................................ 10
  3.6 Talking to CVS .................................................................................................... 10
  3.7 Platform independence ....................................................................................... 11
  3.8 Example usage .................................................................................................... 11

4 Discussion .............................................................................................................. 13
  4.1 C++ API .............................................................................................................. 13
  4.2 MEL ..................................................................................................................... 13
  4.3 Editing ................................................................................................................ 14
  4.4 Unsolved problems .............................................................................................. 15
    4.4.1 Tags ................................................................................................................ 15
    4.4.2 Parent folder ................................................................................................ 15
  4.5 Further development ......................................................................................... 15
Integration of Version Control in Digital Content Creation

4.5.1 Handling of textures.................................................................15
4.5.2 Recent Files...........................................................................16
4.5.3 Preferences..........................................................................16
4.5.4 User interface improvements................................................16
   4.5.4.1 Browser Window............................................................16
   4.5.4.2 History Window..............................................................17
4.5.5 CVS exit codes.......................................................................18
4.5.6 CVS interaction.....................................................................18
   4.5.6.1 API vs command line...................................................18
   4.5.6.2 Pserver vs command line..............................................18
4.5.7 Major feature additions.......................................................18
4.6 Conclusions.............................................................................19
   4.6.1 Testing.................................................................................19
   4.6.2 Plugin development..........................................................19
5 References................................................................................20

Appendix A - Design.................................................................21
Appendix B – Use cases.............................................................22
   Open.........................................................................................22
   Save.........................................................................................24
   Save As.....................................................................................24
   New/Edit project.......................................................................25
   Thoughts...................................................................................26
Appendix C – Button matrices....................................................27
   Up-to-date................................................................................27
   Unversioned.............................................................................27
   Missing.....................................................................................28
   Removed..................................................................................28
   Modified..................................................................................28
   Add.......................................................................................28
   Conflict...................................................................................29
   Resolved................................................................................29
Appendix D - Required modifications of Maya 6.0.1's own scripts......30
1 Introduction

This section introduces the problem which the work behind this thesis tries to solve. It describes the requirements for the solution as agreed upon through discussions at the workplace from which the idea originated. The later part explains the style used in the document and some of the terms and abbreviations used.

1.1 Problem statement

Investigate ways to construct a Maya-plugin for version control. This plugin should integrate version control directly into the main application's interface. Ways to interact with the main application and the version control systems should be explored. A prototype of the plugin with limited functionality should be built.

1.2 Background to the problem

1.2.1 Performing DCC with Maya

Multiple CG-artists are often involved in the same projects, working simultaneously in the same or closely related areas of asset production. Computer games, console games, and CG-animated films consist of multiple levels and/or scenes. Files containing these scenes to a large extent reference other files containing the individual 3D-objects or surface textures. This makes it easier for multiple persons to improve on the same scene in an asynchronous fashion through working on the referenced files.

Maya is an application mainly used for creating and manipulating 3D objects and animations. The actual Maya files are called scenes, regardless of whether they contain only one object or three thousand objects.

1.2.2 Version control

Employees at the company Data Ductus have on several occasions discussed the importance of version control during computer graphics productions. Version control systems simplify the chore of keeping track of changes to the digital assets, file distribution and monitoring of the progress of the project.

To facilitate the use of it in the production pipeline, the idea was to construct a plugin for the main application for many digital artisans, Maya. In that way users are more probable to keep their work in sync with the central file repositories, and have the ability to change between all versions of a file without leaving their creative environment.

1.3 Requirements of the prototype

The File menu should be modified in the following way:

- New Scene – If there are unsaved changes in the current scene a dialogue should be displayed allowing the user to save current work.
- Open Scene – Opens the files which are located in the projects CVS workspace on the local disc.
Integration of Version Control in Digital Content Creation

Introduction

• **Save Scene** – Save in workspace and add file to repository or commit changes.
• **Save Scene As** – Nearly the same as Save Scene.

In the windows for **Open** and **Save** (whose shared features I have chosen to call the **Browse window**) there shall be ways to:

• Update the currently viewed files from the server.
• Toggle between seeing all versions of the files at once or just the latest one.
• Modify tags on files.
• Read commit-logs.
• Enter commit messages when saving.
• Perform some sort of locking to prevent users from editing the same file and thereby creating conflicting changes.

Later versions with full functionality could be considered marketable, so staying clear of license restrictions is preferable.

### 1.4 Stylistic Remarks

• Words highlighted with *italics* are usually the names of GUI-controls.
  
  Example: The menu entry is called **Open**.

• Words highlighted with *monospace* font are either pieces of code, console commands or file system identifiers.
  
  Example: Submit the contents of the current directory with **cvs commit**.

### 1.5 Word list

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>Computer Graphics.</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System, see section 2.1.3 for description.</td>
</tr>
<tr>
<td>DCC</td>
<td>Digital Content Creation.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface.</td>
</tr>
<tr>
<td>Maya</td>
<td>A software program for DCC produced by Alias.</td>
</tr>
<tr>
<td>MEL</td>
<td>Maya Embedded Language, see section 2.1.1 for description.</td>
</tr>
<tr>
<td>stdout</td>
<td>“Standard output is the output stream into which data are written from a program. Data written to standard output are usually written to the screen unless redirected.” [1]</td>
</tr>
<tr>
<td>Subversion</td>
<td>An open source version control system competing with CVS. It is a much younger project and claims to correct many of CVS' faults.</td>
</tr>
<tr>
<td>Version control system</td>
<td>At its simplest, a system for keeping a database of files in multiple versions.</td>
</tr>
</tbody>
</table>
Workspace  The files (usually stored in an isolated tree of directories) which have been transferred from the server to the user's own computer and stored there for having work done on them.
Integration of Version Control in Digital Content Creation

2 Methods

This section describes the tools and methods used during the production of the plugin. See section 4 for additional information about the reasoning leading to the choice of certain tools.

2.1 Tools

2.1.1 MEL

Maya uses its own language [2, 3] to construct its user interface and provide easy access to scene objects without the hassle of a compiled language. This Maya Embedded Language uses a C-like syntax and comes with most standard scripting language features. As a result of its field of use, it is packed with functions for modelling, animation, effects and rendering. I have however limited my focus to functions providing construction and manipulation of the user interface.

2.1.2 Maya C++ API

MEL does provide speedy development and lots of functionality. It does not however provide a way to make plugins or great speed of execution. For this you have to use the C++ API [2, 3].

The API for version 6.0 was chosen over the version for 6.5 (the latest version at the time of writing) because other tools and plugins for Maya are usually built for 6.0 at the time of writing.

2.1.3 CVS

CVS is probably the most used version control system today, it is also what the employees at Data Ductus are used to; it was therefore chosen as the system to build the prototype for.

An effort was made to only include version control system-specific interaction in an isolated part of the plugin. To create interfaces agnostic to the version control system the features of CVS were primarily compared to those of one of its alternatives, Subversion.

2.2 Development process

Beyond standard practices of software development, no specific set of methods or rules have been dogmatically applied during development.

2.2.1 Research

Research was done into these areas:

- The tools used, explained above.
- Features of competing products, primarily NXN Alienbrain, the extent and methods of integration with Maya.
- Some of the ways in which WinCVS (a common CVS-client on the Windows platform) interacts with the CVS console client.
2.2.2 Customer/user discussions

During any software development the end user should be taken into account. Through meetings, semi-spontaneous discussions and inquiries [4] the software can be shaped into a more usable form. Having the privilege of working in an environment in which the users work has been of great help in the design phase of the project. It has also been useful when discovering subtle details which appeared during implementation.

2.2.3 Design

As an aid to identifying problems to be solved and creating a more detailed vision of the final prototype, detailed use cases were written and can be found as Appendix B. Microsoft Visio was used for constructing use case and class diagrams (Appendix A).

2.2.4 Implementation

The implementation working environment consisted mainly of Microsoft Visual Studio .NET 2003, the melStudio plugin for Maya and jEdit. The code itself was never actually version controlled, I didn't deem it necessary since it was to be developed by myself only.

2.2.5 Testing

End user testing of the software system during development is essential for decreasing bugs and other quirks in the final release. See section 4.6.1 for further discussion.
3 Results

Here I present the results of my work, the functionality of the prototype. It begins with the installation procedure and then goes through how things are done behind the scenes and the visible results. At the end an end user example is provided.

3.1 Installation

No efforts were made to produce an installation executable. However it was implemented with caution not to make the procedure unnecessarily complex.

Maya 6.0.1 contains both global and user-specific directories to store plugin-related files. Where one puts the files thereafter depends on whether one wants to share the plugin with other users of the computer.

MAM.mel contains much of the plugin setup and shut down procedures regarding alteration of what functions and commands are called by the user interface. All names of different user interface controls such as menu items and tool bar buttons are stored in this script. Hopefully it will provide some isolation of places which have to be changed for the plugin to function with other versions of the main application. It should be placed in the scripts directory.

MAM.mll contains the compiled C++ code of the plugin. It should be placed in the plug-ins directory.

mam_folder.xpm and mam_genericFile.xpm are used by the browser window and should be put in the icons directory (which in turn is located in the prefs directory for user-specific installations).

A few lines need to be added to two of Maya's own interface scripts, updatePrefWndUI.mel and createPrefWndUI.mel. This is for constructing the preferences controls for the plugin in the preferences window. The resulting modified files are included with the distribution of files. The specific modifications can be found in Appendix D.

3.2 Replacing the calling of original dialogues and windows

3.2.1 File menu

The menu items Open, Save and Save As are modified on plugin load/unload for example on load:

    menuItem -edit -command "mamSaveWindow -incrementalSave" "saveItem";

This results in the Save menu entry calling one of the plugins commands instead of the default one [5].

3.2.2 Main tool bar

The applications main tool bar contains buttons for Save and Open functions. As the modification of their functioning was not really part of the requirements they were overlooked until very late in development. Thankfully the correct behaviour could be provided by a very similar procedure to the file menu entries.
3.2.3 Hotkeys

Hotkeys are in this system's terminology key combinations such as Ctrl+S which perform actions (saves the current file etc.). The functioning of these hotkeys are altered in a similar way to the menu entries and main tool bar.

One thing to be noted is that the Ctrl+S combination is often pressed almost subconsciously while working on a file to ensure that no changes are lost in the event of system malfunction. Therefore it was decided that this action should not bother about version control, but simply behave as normal, just saving the current file to disc.

3.2.4 Unsaved changes

When there are unsaved changes and the user attempts to create a new file, open a different file or exit the application, a dialogue is presented. It gives the user the option of saving her/his changes. If the user chooses to do so, the Save Scene As window comes up for previously unsaved files or the file is simply saved. If the window is displayed the application loses track of what it was doing, otherwise it continues what it was in the process of doing. This forgetfulness works the same in the original and so the priority given to fix it was low. All this is handled through the saveChanges() function located in one of Maya's MEL-scripts.

The scripting language provides the ability to override already declared functions and implement your own instructions to be run instead. I used this feature on plugin initialization thus:

```maya
eval("global proc int saveChanges(string $action)" +
"{ return MAMSaveChanges($action); }";
```

On plugin uninitialization I make the following call to return to default behaviour through having the application re-read the file containing the original function:

```maya
source "saveChanges.mel";
```

3.3 Browser window

The Open Scene window and the Save Scene As window are very much alike and I have chosen to call that foundation of both windows the browser window.

An effort was made to make it behave and look in a not painfully different way to the standard Maya open/save windows which it replaces.

The state of the files in relation to the version control system is computed in the same way as WinCVS does it, through parsing the local Entries file in the CVS directory. This requires the user to have checked out a version of the CVS module's files.

I made the ListView class to support the sorting of the listed files by pressing the column headers. It is also used by the History Window by the way.

A combo box in the browser windows allows the user to specify which file types should be displayed.

The Update button updates the contents of the current directory from the server if no files are selected. If a file is selected, it updates only that file.

The Show History button brings up the History Window to show information on any
selected file.

3.3.1 Open Scene Window

Special features of the open window are the Edit and View buttons.

*Edit* opens the file in editing mode, displaying dialogues if appropriate, depending on the state of the selected file. Matrices of possible actions the user can take depending on the local and remote state of the file were made and are presented in Appendix C. An attempt is made to mark the file for reserved editing.

*View* opens the file for quick viewing. It does not mark it for editing. This button is greyed out if the selected file is not readable. This is likely to occur if the file does in some sense exist according to the version control system but not according to the file system.

![Screenshot of the Open Scene Window.](image)
3.3.2 Save Scene As Window

When selecting a file in the listing, whatever is present in the file name edit box is replaced with the selected files name, and focus is set on the edit box.

When the **Save** button is pressed an attempt is made to save the file and if the action is successful the user is presented with the same set of dialogues as mentioned in the **Edit** case in section 3.3.1. If a file with the same name already exists the user is first asked whether she/he really wants to overwrite it or save it in another name.

3.4 History Window

The original requirement of having the ability to show all versions of all files was altered to include this additional window instead. It provides the user with information on who created what version of a file, when they did it, what they had to say about the changes they made and whether that version is tagged.
3.5 *Preferences Window*

The plugin alters this window's *Files/Projects* section to include settings of its own. When the plugin is not loaded the controls are disabled.

As described in section 3.2.4, MEL does provide you with the option of overriding functions to perform your own instructions. This method was tried out for altering the *Preferences Window* also but with varying results. It seems that the scripts containing the code for constructing that part of the user interface are sometimes read after the plugin has been loaded, resulting in them overriding any functions the plugin has tried to override. Currently the integration is done through modification of Maya's own scripts (as noted in 3.1 and discussed in 4.5.3).

3.6 *Talking to CVS*

I hoped to find a C API for interacting with the CVS server. Such a library is under way in the form of LibCVS [6]. It consists of a main specification accompanied by implementations in various languages.

The C implementation had only got as far as establishing a connection at the time I researched it, so it was ruled out as an option. The implementations in other languages (Perl, Java, Objective C) were more complete. They were ruled out because of the complexity of mixing three different languages in one solution, and the uncertainty of which license the plugin would be released under.

The LibCVS homepage indicates it is released under the GPL though for example the
Perl implementation contains the statement:

“This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.”

I chose instead to use the CVS client executable installed together with GUIs like WinCVS. It is also under the GPL but keeping the code in its own isolated binary and executing separately from the plugin process does, according to the company’s license monkeys, not require the plugin to be put under GPL (or a more liberal license).

The CVS client is a console application from which I was eventually able to get output from, into my application. This was first done in MEL using the `popen` function after reading instructions by Robert Bateman [5]. After that the original C- version of `popen` was tested but failed to work since Maya is a Windows application and not a console ditto. Instead I implemented the solution through executing a child process and retrieving the output by using functions from the Microsoft Platform SDK.

### 3.7 Platform independence

During the development of the plugin, the ability to port it to different platforms was kept in mind. As stated in the previous subsection, in order to retrieve console output I was forced to use Microsoft platform specific code. Supplementing it with new code for other operating systems should be a more straightforward process. It should be possible in that case to use the `popen` function with friends, which seems to be de facto standard at least on UNIX-like platforms.

The low level file system interaction will need modification but has been isolated to the file system class so it should not be that much of a problem.

### 3.8 Example usage

1. After starting Maya the user presses the Open button in the tool bar.

2. The Open Scene window is displayed showing the contents of the scene directory in the latest project the user worked on. The user presses Update to make sure the files are up to date with those on the server.

3. It turns out someone has forcefully modified a file which the user had marked as being edited and modified herself/himself without submitting the change. This results in the file's state changing from “Modified” to “Conflict”.

4. The user selects the file and presses Edit.

Christopher Bergqvist  
Luleå University of Technology
5. A dialogue comes up notifying the user of the conflict and giving the user five options:
   • Just ignore – Opens the file regardless of its state according to the version control system.
   • Force creation of a new version – Creates a new version of the file with the users own modifications regardless of what has been submitted to the server before.
   • Rename the file – Renames the file to avoid the conflict.
   • Replace local version with server version – Self descriptive.
   • Cancel

6. The user chooses to rename the file. The reasoning is to save it under a temporary name in order to compare its state to the state of the file on the server.

7. A dialogue is shown asking the user for the new name. The user enters the name and confirms it by pressing the Rename button in the dialogue.

8. A file with the entered name already exists, so the user is asked whether he/she wishes to overwrite that file.

9. The user chooses to not overwrite the file and is again prompted for a new name.

10. This time the user enters a name which no file has already got, neither locally nor on the server.

11. The user is questioned whether the file should be submitted to the server. The user chooses not to (it is only temporarily saved for the sake of comparison).

12. The file is opened in Maya and the user moves the camera around looking at the progress he/she had made.

13. The user presses the Open tool bar button again in order to view the original file according to the server.

14. Since the user has modified the viewing angle of the camera amongst other things, the scene is treated as changed and a dialogue comes up to ask the user if those changes should be saved. The user chooses to ignore them.

15. The Open Scene window is shown anew.

16. The user chooses to open the file in conflict again, this time choosing to retrieve the server version instead of renaming the local one.

17. The server version of the file contains massive changes, and the user decides to redo her/his own changes.

18. The user presses the Save button in the tool bar.

19. The file is saved locally and then the user is asked if a new version should be created. The user chooses to do so.

20. A dialogue is shown providing the user with the opportunity of entering notes about the changes made. The user makes a note about the changes and presses the Commit & Unedit button.
4 Discussion

The discussion section begins with a discussion on the programming experience. It continues with background on the important topic of marking files for editing and how I solved it. After that come the unsolved problems section, further development (user interface, features, CVS interaction) and conclusions.

4.1 C++ API

Watch out for memory leaks in your plugins! Maya defines two functions, initializePlugin() and uninitializePlugin() each of which seem quite self descriptive. What I discovered halfway through development though, was that uninitializePlugin() is only called if the user chooses to unload the plugin via the Maya interface, not when Maya shuts down. Hopefully the operating system should handle deallocation of unfreed memory on application shut down but if you have any other resources to clean up, this could create a major problem... if there wasn't a nice and clean solution.

The way I did this was by telling Maya to notify me when it shuts down like so:

```cpp
callbackId = MSceneMessage::addCallback(MSceneMessage::kMayaExiting,
onMayaExit, NULL, &result);
```

And then just have onMayaExit() call uninitializePlugin() with correct parameters:

```cpp
void onMayaExit(void*)
{
    uninitializePlugin(MFnPlugin::findPlugin(Plugin::STR_NAME));
}
```

I was a bit disappointed to learn about the fact that the C++ interface was not able to interact with the Maya GUI at first, but I found peace as can be seen in the next section. Development with the API was quite painless, Maya does take some time to load every time you want to start a debugging session but that can't be helped.

Apart from some inheritance of Maya classes, nearly all API-interaction is abstracted into the MayaInterface namespace. This should facilitate support of future versions of the API.

4.2 MEL

I was minding my own business, giving the browser windows the feature of being able to change the current directory when any of the currently displayed directories was clicked on. For some reason Maya kept crashing on me whenever I tried clicking on a directory. Changing current working directory through other ways did not result in a crash. Luckily I had read about something like this [3] just a few nights earlier. The solution was to use the evalDeferred MEL command which waits for inactivity before executing the command passed to it. Turns out I was deleting a UI control while it was being pressed and that did not work well with how Maya does it.

There are little things I miss like replacing one pattern in a string with another in a sane way. I'm glad to have had the C++ interface to support me at those times.

It should be noted that at first I was unsure whether I should use MEL, the possibility is there to use other GUI toolkits as long as they can interface with C/C++. But
wanted to give MEL a chance and in the end I'm quite happy MEL was used for producing the interface. The user interface improvements noted in section 4.5.4 should be quite easy to make. Adding the feature of different icons for different file states could not be made easier, and would probably require lots more work if one wanted to have a fully functioning file view with the help of the UI toolkit MFC. It's very nice to be able to prototype GUIs in an interpreted language. Also, being proficient in MEL is not a bad merit, although using it to interact with the actual scene data has not been done much.

The C++ Standard Template Library's `stringstream` classes were used in a satisfying way to concatenate series of MEL commands across C++ loops and methods. It simplified conversion of numeric values into string format and significantly reduced the number of API calls to execute MEL commands. Compared to having these API-calls everywhere MEL is inlined in the C++ code this probably resulted in some gain in speed.

The action of porting the plugin to other DCC-applications would definitely require a fair amount of work. MEL commands are scattered across the user interface related classes and would require a bit more abstraction to make the creation and interaction with the user interface application-agnostic.

### 4.3 Editing

The ability to mark files as being edited came as a bit of an afterthought. This feature of CVS and other version control systems I dare say is not used by most software projects. This has to do with the fact that traditionally, the users of the system have been programmers whose code is saved in plain text files. If two users edit the same file but in different places, CVS automatically takes care of the merging of the changes into a new version of the file. Even when the software is not able to merge the changes, manually doing so is not so complicated.

Most files digital artists work with are not stored in plain text format. An image for example may contain a header in a predefined format, and then just a sequence of colour values, often in compressed form. Any numbers in the file equalling the value of the line break character are incidental and so merging changes in such files is not possible with most version control systems today.

Because artists commonly don't have the luxury of relying on the software to take care of merging, it is of great importance that they do not modify the same file at the same time. Here, the ability to mark files for editing is essential.

If the `cvs commit` command is executed when a file is marked as edited by the current user, the file is automatically unedited. That was exactly what happened when a user chooses `Edit` in the `Open Scene` window, works on a file, saves the changes and commits them as a new version. If the user continued to modify the scene it was done so without being marked as edited and another user might start editing the same file. This is not a healthy situation. It was remedied at the last minute by modifying the commit window to include a `Commit & Unedit` button. Pressing this new button will carry out the old behaviour whereas pressing the old `Commit` button results in `cvs edit` being run after committing.
4.4 Unsolved problems

4.4.1 Tags

I prioritized away the requirement of having the possibility to modify tags from within the plugin, since I deemed it less urgent than much else.

Another thing concerning tags was the idea and semi-official requirement of treating certain tags as special. The tag “FINAL” should be interpreted as if there should be no more changes made to the file, and the plugin should have complained to the user if an attempt was made to create a subsequent version of a file with such a tag on it.

Both these additions of functionality could probably be dealt with fairly fast though and would have been implemented had I not run into more pressing fixes during the later stages of the project.

4.4.2 Parent folder

When using the browser windows and clicking the parent folder (..) it remains as the selected folder in the new file listing. This makes it difficult to continue browsing to the next parent. One workaround is to select a file and then re-select the parent folder. Another is to use the location controls at the top to select the folder to view when needing to browse up several levels.

Several attempts to correct this problem were made but without success.

4.5 Further development

4.5.1 Handling of textures

At one of the first meetings about the plugin at the company, the problem of managing textures used in scenes was discussed. Usually textures can be shared across several scenes and these scenes can be produced by different people. As one person completes the texture it is updated in all the scenes that reference it. But what if the texture in the process of being adapted to one scene, diverges from matching the look of another one?

At one point in development I believed there would be time to implement the management of textures in the scene. I did some research on it (other users scripts [7, 8], the MEL scripts that drive Maya’s own interface) and came up with these things:

- After opening a scene, a list of names of texture objects can be obtained through the MEL command:
  ```mel
lst -type "file";
  ```
- The file names are retrieved through:
  ```mel
  getAttr objectName.fileTextureName;
  ```
- Once you have the file name of a texture, check to see if it is versioned and if there are newer versions on the server. If it is so, run something like:
  ```bash
  cvs update -p -r 1.3 tarmac.tga;
  ```
  Where 1.3 is the latest version of the file. This will pipe its contents to stdout
instead of into the current file on disc. This piped data can then be saved to a temporary location.

- It took more than one day to find it, but one possible way to display textures two dimensionally and resized to specific dimensions is to use the `swatchDisplayPort` command.

```csharp
string $shader = `shadingNode -asTexture "file"`;
string $pathAndFileName = "D:/Maya/Project/textures/tarmac.tga";
setAttr -type "string" ($shader + ".fileTextureName") $pathAndFileName;
swatchDisplayPort -wh 64 64 -sn $shader;
```

Using the `swatchDisplayPort` does as shown above require the creation of a new shadingNode in the scene. This node should be deleted by the plugin when no longer useful. One irritation with doing it by creating new nodes is parts of Maya's interface directly respond to it, like the Attribute Editor. An alternative way would be to use the C++ API to create a custom control for displaying textures without creating any nodes to hold them. This possibility was not explored.

### 4.5.2 Recent Files

One menu entry which was overlooked during the establishment of requirements was the `Recent Files` one. Files opened through this menu are not immediately checked against their state according to version control. Once a file is opened the user can issue a `Save` and then be prompted with eventual warnings.

This should probably be one of the most prioritized areas of improvement as it is so frequently used.

### 4.5.3 Preferences

In part I am dissatisfied with my results, the plugin requires that two of Maya's own MEL files are altered in order to include the preferences settings for the plugin. The solution gives the end user a greater sense of integration and the partially competing product NXN Alienbrain also requires modifications of the main application's scripts to be made. However, a cleaner way of doing it would be to provide an isolated settings-window accessible from the browser window.

Because the plugin itself might not be loaded before the user brings up the preferences window, practically all UI-commands related to the plugin are entered directly by the modified scripts. The consequence of this is hard coded version control specific controls, in this case geared towards CVS. Having a separate window which was created through C++ would make for a better solution where different controls could be created depending on the version control system used.

### 4.5.4 User interface improvements

#### 4.5.4.1 Browser Window

- Adding a set of tool bar-like buttons. These could be for functions like:
  - Updating the whole directory.
• Modifying the tag of a file.
• Unediting a file without committing it.

• The plugin could be modified to give users the ability to add shortcuts to applications used to open for example texture files. That suggestion came from Oskar [4] after I added the Editors column, something WinCVS lacks.

• Adding a combo box for quickly changing location to “Current scenes”, “Current project”, “Home” etc. as the original windows have it (Figure 6).

• Adding the column of shortcut icons found to the left in many windows applications today, including the original Maya ones. The information about which icons it should contain could be controlled by the user via some preferences window. The information on the icons which are displayed in standard open/save windows is stored in the Windows Registry, so it should not be that hard to get hold of for the plugin.

• Different icons for representing different file states.

Care should be taken to prevent it from getting as cluttered as the interface of WinCVS though.

4.5.4.2 History Window

A way to limit how far back the history window queries for and displays history was planned as a drop down combo box containing options like “yesterday”, “last week”, “last month” etc. Calculating such relative dates in a correct way and possibly taking into account client/server difference in time was not something I wanted to devote my time to. However, CVS does actually understand many of those relative dates, and if there was some documentation somewhere, the task of compiling a working list for CVS would be simple. If no documentation can be found, maybe the CVS source
code could be looked into for clues. No research has been made into any other version control system's ability to handle relative dates so it might have to be implemented into the plugin.

4.5.5 CVS exit codes

The current code does support the retrieval of exit codes from the CVS console application. However no action is taken depending on the value of these codes. With sufficient knowledge, these codes could enable further user interface enhancements. For example during testing, the plugin just refused to work to its full capacity. This was soon found to depend on the user not having logged in correctly to the server, and so most, if not all CVS commands failed to execute.

The exit codes were not considered during the design phase and I did not have enough time during implementation to research what codes are returned on which occasions. No detailed documentation on this seems to exist. One could research this further, read the source code, contact developers, but I did not have the time.

4.5.6 CVS interaction

4.5.6.1 API vs command line

Communicating with CVS through parsing console output does not result in the most elegant solutions. Output from different versions of client/server combinations may have discreet variations in format. This should through my design only leave the plugin with less information, not incorrect information. Having a C/C++ API to access the version control system is preferable though, and if one should want to add a Subversion interface to the plugin this possibility should definitely be explored.

4.5.6.2 Pserver vs command line

A third possibility also exists, to communicate to the CVS server directly using the Pserver protocol. The plugin would then become the actual client with all the responsibilities that it includes:

- Setting up a network connection.
- Interpreting and sending correct Pserver commands.
- Management of bookkeeping and version controlled files on the client filesystem.

Doing it that way would result in a much more complex solution. If the requirement of encrypted communication was added for example, a security API would probably need to be integrated. Accessing the available CVS command line client would probably lend that functionality without much hassle. This is more in accordance with the “small is beautiful” philosophy of UNIX development, instead of building a monster; use many small pieces that individually do one job and do it well.

4.5.7 Major feature additions

There are a couple of ideas planned for a final release version of the plugin, namely integration into the management of references in the scene and full version control of nodes in the scene graph. Import and maybe export action could also be enabled to
4.6 Conclusions

4.6.1 Testing

The software was installed on three end user computers and I held a meeting with the users about the plugin's functionality and working with CVS in general. The drawback is testing began rather late as it took some time to get the plugin up to a usable level. By that time they were spending much of their time writing their own thesis papers. But the testing that was performed resulted in several improvements.

Building a test environment for the plugin would be good, since I am sure some of the code has never had the chance to be tested at runtime. One could create fake files and a fake CVS client to make sure all cases are tested, but knowledge about the real system's behaviour would be required. Maybe creating a script/application which creates a temporary CVS module and sets it up in a way that as many cases as possible can be tested would be the best way. Some cases are very hard to test though, as they occur when two users are interacting with the server at virtually the same time.

4.6.2 Plugin development

This was my first attempt at plugin development and also my first real experience of working with Maya from a programmer's perspective. At first there was some traditional moping about how “it's a miracle that their system actually works”. But after a while it starts to make sense and you get a feeling for the reason behind the former mystery of their success. It has brought me insight into how truly extensible applications can be built and for that I am grateful.

The work done here will hopefully come to ease the work of CG-artists, letting them concentrate more on being creative, not having to worry too much about the robotic chore of the version control business.

There are ideas of making it into an open source project and Data Ductus are positive about that. However the license costs for acquiring Maya are currently a bit of an obstacle for the original author of the plugin. Regardless of the development method it is my belief that the prototype does provide future developers with a firm ground for further development and major feature additions.
5 References


[4] Holmstrand, Oskar, Computer Graphics student at Luleå University of Technology's institution at Skellefteå. primal@tpdata.com


[6] LibCVS.
https://libcvs.cvshome.org (2005-06-03)


[8] HIGHLAND3D.COM.
The UML class diagram above shows all the classes, namespaces and structs included in the Maya Asset Manager plugin prototype. The `CmdPreferencesWindow` class is greyed out to indicate it was not used in the final version of the prototype.
Appendix B – Use cases

This document was written before implementation and things did not all turn out to be done exactly as detailed here. It contains a few (at the time of writing) unanswered questions marked with bold type.

Open

Preconditions

The user chooses Open Scene from the File-menu or presses Ctrl+O.

Flow

1. A window is displayed. It contains:
   - A list of files in the current directory. The directory is either the previously viewed one or the projects scene-directory. The list contains columns for file name, version, tag and status (Up to date, Modified, Conflict, Missing, Unversioned). It is possible to sort the files by each column in ascending or descending order as in Windows Explorer for example.
   - An Update button for updating the files in the current directory from the version control system (cvs update). The update is not allowed to recurse into child directories because that may cause problems with texture management and so on.
   - The Update button is accompanied by a pair of radio buttons with the descriptions:
     - Update all files.
     - Update only files I have not reverted.
     Maybe these could be moved away under a Settings-menu.
   - A button for bringing up a dialogue for choosing the directory to be displayed.
   - A text field where the current path is displayed. This field has a popup menu which contains parent-paths to enable the user to quickly navigate up towards the file system root.
   - A combo box where one can choose the search criteria for the files to be displayed: Maya binary (*.mb), Maya ASCII (*.ma) or Best Guess (*.*). Should the be more file types to choose from?
   - A Modify Tag button for bringing up a dialogue making it is possible to change the tag of the selected file.
   - A Open button for opening the selected file.
   - A Cancel button.

2. The user chooses a file from the list through clicking on it and then on the Open button.

3. One of four cases occur depending on the status of the file:
Integration of Version Control in Digital Content Creation   Appendix B – Use cases

a) The same version of the file is already located on the server, everything is okay.

b) There are local modifications to the file which have not been sent to the server (or the file simply doesn't exist on the server at all). A dialogue is displayed asking if the user wishes to send of the changes and create a new version before opening the file for editing. If the user chooses to submit the file problems can occur if someone else happens to send away a new version prior to the current user. How should this exceptional condition be resolved? Checking of exit code from the CVS-client?

c) The selected file has been marked as in conflict. A dialogue is displayed with the following alternatives:

- **Save local file under new name.**
- **Discard my own changes.**
- **Force creation of new version based on my file.**
- **Just open my file! (Not advised.)**
- **Cancel.**

d) The file has been removed from the server. Nearly exactly the same dialogue as in case c is displayed.

4. If a file has been opened all files which are referenced by the file are gone through with `cvs -qan update` (query, only asks about changes without retrieving them). If any of the referenced files have been updated then they are added to a list that is displayed to the user.

When the user clicks on files in the list which are image files then the image is displayed in old and new version.

If the user is dissatisfied with changes made on one or more files these can be marked (every file name has a check box beside it) and then the user can push a button **Save old versions under new name**.

*Are the new versions put in a temporary folder before being displayed?*

5. If the user chose **Save old versions under new name** a dialogue is displayed with a list of the marked files with the columns:

- **Old file name**, contains fields with all the old names.
- **New file name**, contains editable text fields where the user can fill in the new name.

Besides that there are probably only the buttons **OK** and **Cancel**.

**Postconditions**

A new file has been opened or the user has aborted and the currently opened file is left open.
**Save**

**Preconditions**

The user has a previously saved file and does one of the following:

- Has unsaved changes and selects *Open Scene* in the *File* menu or presses *Ctrl+O*.
- Selects *Save Scene* in the *File* menu or presses *Ctrl+S*.

**Flow**

1. A dialogue is displayed giving the user the following alternatives:
   - *Make new version*.
   - *Save locally*.
   - *Cancel*.

2. If the user chooses *Make new version* the status of the file is checked against the CVS server. If there is a conflict a dialogue with these alternatives is displayed:
   - *Save As new file*.
   - *Force creation of new version based on my file*.
   - *Cancel*.

   If alternative b is chosen and the file is tagged with “FINAL” in its current version a warning dialogue is displayed to alert the user of that fact.

3. *Make new version* (from 1) leads to yet another dialogue similar to the one displayed when attempting to open a modified file in the “Open” use case above.

**Postconditions**

Either the user has aborted and it is as though nothing had happened or the file has been saved. If the later case is true and the use case was activated through an Open operation then that window is displayed.

**Save As**

**Preconditions**

Any of the following:

- The user selects *Save Scene As* in the *File* menu or presses *Ctrl+Shift+S*.
- The user has an unsaved file and selects *Save Scene* or *Open Scene* in the *File* menu or presses *Ctrl+S/Ctrl+O*.
- A conflict has occurred when the user has attempted to save a file and the user has chosen to save the file under a new name to avoid the conflict.

**Flow**

1. A window is displayed. It is almost the same window as in the Open case with the
list of files, the text field displaying the current path, the combo box to select shown file type and buttons to select directory, update the files and cancelling. The only thing that is changed is that the Open button has been replaced by a Save button and that there has been added a text field for entering the new file name.

2. The user can interact with the GUI in about the same way as in the Open case. Clicking on a file in the list results in the contents of the text field for the new file name being replaced with the selected file's name.

3. The user selects a file in the list and presses the Save button.

4. If there already exists a file with the given name the user is presented with a dialogue containing the alternatives:
   - Create a new version of the old file containing seemingly new data.
   - Cancel.

5. If the user hasn’t aborted earlier the file is saved to disc and cvs add and cvs commit or just cvs commit is run.

**Postconditions**

The file has been saved or the user has aborted and it is as if nothing ever happened.

**New/Edit project**

**Preconditions**

The user chooses New or Edit Current in the File -> Project menu.

**Flow**

1. A modified version of the project settings window is displayed. The new thing about it is a section on CVS settings. In this section there is:
   - A check box for choosing whether the version control system integration is activated.
   - A text field with the file system path to the CVS console client. A browse button is also provided.

Really, the above are things which can pose to be problems if stored in the project, therefore they are probably saved together with Maya’s global settings.

- A text field containing CVSROOT (right now there are several but I am not really sure how to parse it so it could be better just to leave it as one field).

- A text field containing the name of the CVS-repository.

The information in the two fields above is read from and written to the files projectDir/CVS if they exist.

2. The user changes the settings in the manor she/he wishes.

3. The user presses the Cancel or Accept button.
Postconditions

If the user exits through some other way than pressing Accept (Save in Preferences) nothing is changed. Else:

The status of the check box is saved together with the rest of Maya's global preferences (with the optionVar command). How is change of CVSROOT to be managed? cvs update -qn after?

Thoughts

The questions is whether all settings sans CVSROOT should be moved the Preferences window instead.
Appendix C – Button matrices

As prototyped in the use cases, a myriad of dialogues have to be in place to deal with all the states a file can be in. Depending on a situation, it is suitable to limit the number of actions a user can take to resolve it. This is done to make the dialogues less of an obstacle. File state according to the local bookkeeping files the CVS client creates are represented by each heading. Actions the user can take (the caption of buttons presented) are listed in the leftmost column. Server side states are listed horizontally. A one (1) signifies that the button should be shown when the local and server states are true. A zero means that the button is not really relevant for given states.

This document is merely provided to give insight into one of my efforts to organize development when faced with a large problem. The actual implementation may differ in a few ways.

### Up-to-date

<table>
<thead>
<tr>
<th>Button action</th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Unversioned

<table>
<thead>
<tr>
<th>Button action</th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Missing

<table>
<thead>
<tr>
<th></th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Removed

<table>
<thead>
<tr>
<th></th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Modified

<table>
<thead>
<tr>
<th></th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Add

<table>
<thead>
<tr>
<th>Action</th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Conflict

<table>
<thead>
<tr>
<th>Action</th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Resolved

<table>
<thead>
<tr>
<th>Action</th>
<th>Unable to get server info</th>
<th>File exists on server</th>
<th>File has existed</th>
<th>File never existed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just ignore</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Force creation of new version</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rename the file</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Replace local version with server version</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve file from server</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Retrieve version prior to removal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Submit file to server</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resurrect file</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Undo removal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Appendix D - Required modifications of Maya 6.0.1's own scripts.

- In `createPrefWndUI.mel`, the following modification should be made:

```mel
global proc prefsCreateFileIO()
{
    // Maya's own code
    //////////////////////////////////////////////////////////////////////
    // Maya Asset Manager Code Begins
    //////////////////////////////////////////////////////////////////////
    int $mamLoaded = `pluginInfo -q -loaded "Maya Asset Manager"`;
    separator -style "none" -h 5;
    frameLayout -l "Maya Asset Manager Settings";
    columnLayout -adj true -cat "left" 35 -cal "left";
    text -l "CVS Command Location";
    columnLayout -adj true -cat "left" 40 -cal "left";
    rowLayout -nc 2
        -cw2 300 50
        -cal 1 "both"
        -cal 2 "left"
        -ct2 "both" "left";
        string $mamTextField1 = `textField
            -enable $mamLoaded
            "mamCVS_commandField"`;
        textField -e
            ("MAMUpdatePref("mamCVS_command", "" + $mamTextField1 + "")")
        $mamTextField1;
        symbolButton -enable $mamLoaded
            -image "navButtonBrowse.xpm"
            -c ( "fileBrowserDialog -m 0 -an "Select_Exe"
                 "mamCVS_commandButton";
    setParent ..;
    setParent ..;
    separator -st "none" -h 4;
    text -l "CVS User Name";
    columnLayout -adj true -cat "left" 40 -cal "left";
    rowLayout -nc 1
        -cw1 300
        -cal 1 "both"
        -ct1 "both";
        string $mamTextField2 = `textField
            -enable $mamLoaded
            "mamCVS_userField"`;
        textField -e
            ("MAMUpdatePref("mamCVS_user", "" + $mamTextField2 + "")")
        $mamTextField2;
    setParent ..;
}
```

Christopher Bergqvist
Luleå University of Technology
Christopher Bergqvist
Luleå University of Technology