Version Control in AToMPM - Requirements

Simon Van Mierlo

This document describes the requirements of a version control system as it would be implemented for AToMPM. The system is responsible for tracking versions of a model-driven engineered project, consisting of formalisms, models and transformations. It consists of the following subsystems:

- The AToMPM system, which remains unmodified. This system has a client-server architecture: the user edits artifacts of the project inside of a browser and saves these on the server. The server is also responsible for running transformations. A change which has to made is that all edit operations have to be logged explicitly. Now, they are logged on the AToMPM server to allow for re/undoing changes, but the log has to be saved alongside with the actual artifact. This should be done when the user saves the artifact to disk.
- A version control system, which is to be created. The system (which runs on a server) contains a number of repositories, each corresponding to a version-controlled project. Each project and its artifacts have an edit history associated with them and it is the task of the version control system to keep track of this history and allow the user to revert his or her local copy of the project back to an earlier version, or view the differences between two versions of the project.
- An AToMPM plug-in, whose task it is to communicate with AToMPM on the one side and the version control system on the other hand.
- A conflict-resolution environment. Whenever the local, working copy of a project is updated, remote changes which are stored in the version control system are merged with local changes which were not yet stored in the system. When this merging cannot be completed automatically, the user has to manually intervene. To this end, an environment has to be created which shows the user which parts of the artifact are in conflict.

In the rest of the document, an exhaustive list of use cases will be described.

1. Committing a Project

1. The client chooses a folder, which is the root folder of the project, to be committed to the version control system.
2. The location of the folder is sent by the client to the AToMPM version control plug-in.
3. The plug-in checks whether the project is already version controlled.
    a. If it is, the following steps are performed.
        i. The changes which have been performed by the client on each artifact are retrieved by the plug-in.
        ii. These changes are sent to the version control system, together with the version number of each artifact which is to be updated.

iii. For each artifact to be updated, the version control system compares the version numbers which it received to the version numbers which are stored in its system.
    1. If one or more of the version numbers which the version control system received is smaller than the corresponding version number in the system, an error is returned.
    2. If all version numbers which the version control system received are equal to the corresponding version numbers in the system, the following steps are performed.
        a. The changes are stored appropriately in the system.
        b. A success response is returned.
        c. The plug-in removes the local changes, which are no longer needed, as they are stored in the version control system.
iv. The plug-in relays the response to the client.

b. If it isn't, the following steps are performed.
    i. The plug-in retrieves a list of all current repository names in the version control system.
    ii. The list is presented to the user, and is requested to enter a unique, new name for the new repository.
    iii. All artifacts belonging to the project and the directory structure are retrieved by the plug-in.
    iv. All information is sent to the version control system, which stores an exact copy of the project's artifacts in the specified repository.
    v. The plug-in stores the name of the repository and its current version number.

2. Cloning a Project

1. The client requests a list of repositories from the plug-in.
2. The plug-in relays the request to the version control system.
3. The version control system replies with a list of repositories.
4. The plug-in relays this reply to the user.
5. The user chooses a repository and sends its choice to the plug-in.
6. The plug-in relays the request to the version control system.
7. The version control system sends an up-to-date version of all artifacts contained in the repository to the plug-in. It does this by combining the change sets it has stored with the original versions of the artifacts, calling an appropriate composer which is capable of composing the deltas which were stored with the original versions of the artifact.
8. The plug-in saves the project in a user-specified root folder – which has to be empty – along with the current version of the project and the name of the repository.

3. Updating a Project

1. The client requests one of its projects to be updated.

2. The plug-in extracts the necessary information from the local project: version number and repository name.
3. It contacts the version control system, sending it the repository name and version number.
4. The version control system calculates the changes which were made between the version of the project specified by the received version number and the current version of the project.
5. It sends these changes (or edit scripts) back to the plug-in.
6. The plug-in uses these scripts to update the local artifacts.
7. If a conflicting change is found, the user is requested to manually resolve the conflict, by launching a conflict-resolution environment.
8. The process is complete when all artifacts of the project are updated and the current version number is stored locally.