**Module Specification**

**1. Behavior Hiding Module**

**1.1 Decompositions:**

* **User Interaction Module**
  + **parse :: [String] -> IO String**
    - Reads the list of arguments from the command line and checks whether a command is valid or not.
    - If everything is valid, it passes the command and its arguments to the **postProcess.**
  + **postProcess :: String -> [String] -> IO String**
    - Reads the command and its argument and confirms each command for its syntax such that each command receives at least the number of arguments its supposed to get;
    - For each command, we then invoke the corresponding module from the Functionality module.
    - **Related behavior document:**
      * Behavior-for-Merging.pdf
* **Functionality Module**
  + **Every function (add, remove, …) is a submodule of the Functionality Module.**
  + **add :: String -> IO String**
    - **Uses**
      * Uses **addFile**, **removeFile**, and **getTrackedSet** from the Tracked Set (TS) submodule under Concept Module;
      * Uses interface(s) from the OS hiding module to get the list of files in CD (Current Directory)
    - **Logic**:

CD: List of files in Current Directory

getTrackedSet returns a TS

* + - * When (file not in TS) && (file not in CD)
        + Report error
      * When (file in TS) && (file in CD)
        + Do nothing
      * When (file in TS) && (file not in CD)
        + Call removeFile on file
      * When (file not in TS) && (file in CD)
        + Calls addFile on file
        + Report success
    - **Related behavior document:**
      * Behavior-for-Versioning.pdf
  + **remove :: String -> IO String**
    - **Uses:**
      * Uses **removeFile** and **getTrackedSet** from the Tracked Set (TS) submodule under Concept Module
    - **Logic**:

**getTrackedSet** returns a TS

* + - * When (file not in TS)
        + Report error
      * When (file in TS)
        + Call removeFile on file
        + Report success
    - **Related behavior document:**
      * Behavior-for-Versioning.pdf
  + **init :: IO String**
    - **Uses**
      * Uses **isRepo** and **createRepo** from Repo module.
    - **Logic**
      * Checks if the current directory is already a repository using **isRepo;**
      * If the current directory is not a repo then, we call **createRepo** to initialize an empty repo;
    - **Related behavior document:**
      * Behavior-for-Versioning.pdf
  + **push :: String -> IO String**
    - **Uses**
      * Uses all the submodules, i.e., Communication, Repo, Commit, Tracked Set (TS) submodules in the Concept Hiding Module;
    - **Logic**
      * Push: push from the local repository to the remote one;
      * Checks the validity of the input address;
      * Fetches all the files. If it is a remote address, use the **DownloadRemoteDir** utility in the **Communication** module, and if it is a local address, we will use **copyRepo**;
      * Creates a temporary directory to perform the merging functionality:
        + Checks if the directory is a dvcs project using **isRepo** in the **Repo** submodule. Returns and prompts error if it is not;
        + Checks if the project id of the remote project is the same as the current one. Returns and prompts error if it is not;
        + **Needs to performs pull first if local is not ahead of remote** (so we will delete the current temporary directory and prompt user to call dvcs pull)**;**
        + Performs a logic check to decide whether to do a 3-way merge or a Fast-forward case (details in the Merging Behavior document);
        + Performs the actual merge;
        + Copy the temporary directory back to the remote computer;
      * Deletes the temporary directory;
    - **Related behavior document:**
      * Behavior-for-Merging.pdf
      * Behavior-for-Versioning.pdf
  + **pull :: String -> IO String**
    - **Uses**
      * Uses all the submodules, i.e., Communication, Repo, Commit, Tracked Set (TS) submodules in the Concept Hiding Module;
    - **Logic**
      * Pull: from the remote repository to the local;
      * Checks the validity of the input address;
      * Fetches all the files. If it is a remote address, use the **DownloadRemoteDir** utility in the **Communication** module, and if it is a local address, we will use **copyRepo**;
      * Creates a temporary directory to perform the merging functionality:
        + Checks if the directory is a dvcs project using **isRepo** in the **Repo** submodule. Returns and prompts error if it is not;
        + Checks if the project id of the remote project is the same as the current one. Returns and prompts error if it is not;
        + Performs a logic check to decide whether to do a 3-way merge or a Fast-forward case (details in the Merging Behavior document);
        + Performs the actual merge;
      * Deletes the temporary directory;
    - **Related behavior document:**
      * Behavior-for-Merging.pdf
      * Behavior-for-Versioning.pdf
  + **commit :: String -> IO String**
    - **Uses**
      * Uses all the submodules, i.e., Commit, Repository (Repo) and Tracked Set (TS), from the Concept Hiding module;
    - **Logic**
      * First, checks if the TS is empty. If TS is empty, prompt a message indicating the situation, otherwise, continue the rest of the logic;
        + this part uses the **getTrackedSet** function (in the **TS** submodule), and performs the check;
      * Removes all files that are in the TS but not in the current directory (CD) any more;
        + this part uses the **cleanTrackSet** function (in the **TS** submodule);
      * Gets all the files belonging to the HEAD commit. We name this set PC\_files;
        + this part uses the **getHEAD** function (in the **Repo** submodule);
      * Checks the states of all the files belonging to PC\_files, and those belonging to TS. It should be noticed that the state of a file can belong to one and only one of the following set: {**New** (in TS, not in PC\_files), **Altered** (in TS and PC\_files, with new changes), **Unaltered** (in TS and PC\_files, no new changes), **Deleted** (not in TS, in PC\_files)};
        + this part uses **getCommitFile** function (in the **Commit** submodule);
        + the checking logics will be implemented in the functionality module;
      * Checks following the logic here: (Defer situation) if there are no New or Altered files, prompts a message indicating the situation and returns; (Success situation) else, creates a new commit, prompts a message indicating the situation and returns;
        + this part can possibly use all functions apart from **getCommitFile** (in the **Commit** submodule);
      * Updates the HEAD commit;
    - **Related behavior document:**
      * Behavior-for-Versioning.pdf
  + **clone :: String -> IO String**
    - **Uses**
      * Uses interfaces from the OS hiding module to check if the given path exists locally or not;
      * Uses **copyRepo, isRepo** from Repo module and **InitRemoteConnection, DownloadRemoteDir** in the Communication module;
    - **Logic**
      * If the local path exists, we just call **copyRepo.**
      * Else we call **DownloadRemoteDir.**
      * Checks if the remote “.dvcs” folder exists, and copies the remote directory to the current directory if it does**.**
    - **Related behavior document:**
      * None

**2. Software Decision Module**

**2.1 Decompositions:**

Note that the FilePath and String are equivalent in the following signatures.

* **Utility Module**
  + **DvcsInterface**
    - This module provides the interface to operate the metadata in the .dvcs directory; including path definition and some utility functions.
    - **findDir :: FilePath -> IO String**
      * Uses Haskell library System
      * Returns the first found path for the given file;
      * Obtains the returned String in a do block**, using**  p <- findDir “info”
    - **InsertDirs :: [FilePath] -> FilePath -> IO ()**
      * Copies multiple directories into a given destination path;
    - **copyDir :: FilePath -> FilePath -> IO Exception**
      * Copies a src (the second arg) to a dest (the first arg); i.e. copyDir dest src;
* **Communication Module**
  + **UploadRemoteDir :: String -> IO ()**
    - Uses ssh and scp recursively to copy current folder from the local host server to the given remote server location.
  + **DownloadRemoteDir :: String -> IO ()**
    - Uses ssh and scp recursively to copy a folder from the given remote server to the local host in the current directory.
* **Concept Hiding Module**
  + **Meta Organization Module**
    - Includes the organization details of the .dvcs directory, so that when the paths for metadata need to change, we just only need to modify this file.
  + **Commit Module**
    - Uses the MetaOrganiztion module;
    - **createCommitDir :: String -> IO CommitID**
      * Creates a directory (with the ID as name) and a metadata file (with the input as the message) for the new commit;
      * Returns the randomly generated CommitID;
      * Uses the **createDirectory** provided by the Haskell library System.Directory;
    - **getCommitChilds :: CommitID -> IO [CommitID]**
      * Gets the children of a commit;
      * Uses the Commit module;
    - **getCommitParents :: CommitID -> IO [CommitID]**
      * Gets the parents of a commit;
    - **setCommitChilds :: CommitID -> IO [CommitID] -> IO ()**
      * Sets the childs of a given commit;
    - **setCommitParents :: CommitID -> IO [CommitID] -> IO ()**
      * Sets the parents (using the second arg) of a commit (the first arg);
    - **addCommitChilds :: CommitID -> [CommitID] -> IO ()**
      * Adds children to a commit;
    - **getCommitFile :: CommitID -> String -> IO String**
      * Gets a file belonging to a commit;
  + **Repo**
    - Uses the Commit, Trackset and MetaOrganization submodule (of the Concept Module);
    - **createRepo :: IO ()**
      * Creates '.dvcs' directory and a project metadata file if “.dvcs” doesn’t exist.
    - **getHEAD :: IO CommitID**
      * Returns the commit id for the **HEAD**;
    - **getPID :: String**
      * Returns the project id;
    - **setHEAD :: CommitID -> IO ()**
    - **getLocalLeaf :: IO CommitID**
      * Returns the local leaf;
    - **getRemoteLeaf :: IO CommitID**
      * Returns the remote leaf;
    - **getRemoteHEAD :: IO CommitID**
      * Returns the remote head
    - **getRemotePID :: IO CommitID**
      * Returns the id of the remote project
    - **getRemoteTrackedSet :: IO [String]**
      * Returns the trackedset of the remote project
    - **isRepo :: IO Bool**
      * Checks if the current directory is a valid repo;
    - **insertCommit :: CommitID -> IO String**
      * Inserts a given commit into the current directory;
  + **Trackset** 
    - Uses MetaOrganization module;
    - **addFile :: String -> IO ()**
      * Calls OS hiding module interface to load the tracked set;
      * Adds the file to the tracked set;
      * Calls OS hiding module interface to save the new tracked set;
    - **removeFile :: String -> IO ()**
      * Calls OS hiding module interface to load the tracked set;
      * Removes the file from the tracked set;
      * Calls OS hiding module interface to save the new tracked set;
    - **getTrackedSet :: IO [String]**
      * Calls OS hiding module to load the tracked set;
    - **cleanTrackedSet :: IO ()**
      * Calls OS hiding module interface to load the tracked set;
      * Removes files from the tracked set which are not in CD;
      * Calls OS hiding module to save the new tracked set;

**3. OS Hiding Module**

To fulfill the functionality of the OS Hiding Module**,** we will make use of the utilities in [Hackage](http://hackage.haskell.org/package/directory-1.3.4.0/docs/System-Directory.html).

**3.1 Specifications:**

Below are several system utilities we have used so far in the dvcs implementation.

* [System.Directory](http://hackage.haskell.org/package/directory-1.3.4.0/docs/System-Directory.html): used for file and directory manipulations;
* [System.FilePath.Posix](http://hackage.haskell.org/package/filepath-1.4.2.1/docs/System-FilePath-Posix.html): used for filepath manipulations;
* [Data.Aeson](http://hackage.haskell.org/package/aeson-1.4.6.0/docs/Data-Aeson.html): used for JSON file manipulations;
* [Data.Time](http://hackage.haskell.org/package/time-1.9.3/docs/Data-Time.html): used for getting time and date information;
* [Test.RandomStrings](http://hackage.haskell.org/package/random-strings-0.1.1.0/docs/Test-RandomStrings.html): used to generate random strings;