PUCRS/HP Agreement

LIS – Laboratory of Innovation in Software

**Non-Volatile Memory**

Technical report

TR 2014

Porto Alegre, June, 2014.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Team  Version 1.1  Authors:   |  |  | | --- | --- | | Taciano Perez  taciano.perez@hp.com |  | | Micheli Sielecki  pm.lis@cpca.pucrs.br | Matheus Alves  tl.lis@cpca.pucrs.br | | Pedro Monteiro  pedro.monteiro@cpca.pucrs.br | Natan Facchin  natan.facchin@cpca.pucrs.br | | Cristovam Lage  cristovam.lage@cpca.pucrs.br |  | |  |  | |
| **PUCRS/HP Agreement**  **LIS – Laboratory of Innovation in Software**  Coordination - PUCRS: Prof. Marcelo Hideki Yamaguti, PhD  Projects Manager - HP: Paulo Sá  Technical Leader - Processes - HP: Dante Antunes  Technical Leader - Technologies - HP: Alexandre Soares  Technical Team:  Project Manager: Micheli Sielecki  Technical Leader: Matheus Alves  *Study developed by LIS - Laboratory of Innovation in Software of the PUCRS/HP Agreement, financed by Hewlett-Packard Computers Ltd. with resources of Law 8.248/91.* |



|  |
| --- |
|  |
|  |

Summary

1 Introduction 5

2 Stories and Tasks 5

2.1 Organize Repository 5

2.1.1 Organize Repository - Update configure file 5

2.1.2 Organize Repository - Remove project/configuration files 5

2.1.3 Organize Repository - Reformulate Branches 5

2.1.4 Organize Repository - Search for Ubuntu Git clients 5

2.2 Define and Save Needed Structures 6

2.2.1 Define and Save Needed Structures - Save/Load heap 6

2.2.2 Define and Save Needed Structures - Save utf8/classes hash table 6

2.2.3 Define and Save Needed Structures - Find Needed Structures 6

2.2.4 Define and Save Needed Structures - Load utf8/classes hash table 6

2.3 Organize Development Environment 7

2.3.1 Organize Development Environment - Install native Ubuntu/Eclipse 7

2.3.2 Organize Development Environment - Install Git gui 7

2.3.3 Organize Development Environment - Import current sprint branch 7

2.3.4 Organize Development Environment - Boost VM performance 7

2.4 Create Adapter Class 8

2.4.1 Create Adapter Class - Identify places in code where the team has made changes 8

2.4.2 Create Adapter Class - Create an "Adapter Class" to hold all changes 8

2.4.3 Create Adapter Class - Change code to use adapter class 8

2.4.4 Create Adapter Class - Adapt utf-8 hash table changes 8

2.4.5 Create Adapter Class - Adapt classes hash table changes 9

2.4.6 Create Adapter Class - Adapt heap initialization changes 9

2.4.7 Create Adapter Class - Search generic hash table manipulation functions 9

2.4.8 Create Adapter Class - Search in code calls for hash structures so we can adapt all relevant functions 9

2.4.9 Create Adapter Class - create usage sample for UTHash hash tables 9

3 Tests 10

4 Support Software 11

4.1 Github 11

4.2 UTHash 11

4.3 Eclipse 11

5 Reviews 12

# Introduction

This document aims to provide a high level documentation of the NVM (Non-Volatile Memory) project. In its sections you will find information about the stories and tasks developed during the project sprints. Also, you can verify the testing methods and support software used by the team.

# Stories and Tasks

## Organize Repository

* Update configure file
* Remove project files
* Remove configuration files
* Reformulate Branches
* Search for Ubuntu Git clients

## Organize Repository - Update configure file

We have encountered a problem when trying to run the JamVM without the -O2 optimization flag on the configure file. The solution we opted to use is to manually change the makefiles flags to -O0.

## Organize Repository - Remove project/configuration files

We have deleted all unnecessary files inside our repository and updated the ignore file so no more unnecessary files are committed.

## Organize Repository - Reformulate Branches

We have changed the way we created and used branches. Now we create only one branch per sprint.

## Organize Repository - Search for Ubuntu Git clients

We have decided to use Git gui as a graphical repository management program.

## Define and Save Needed Structures

* Save heap
* Load heap
* Save classes hash table
* Load classes hash table (Nice to have)
* Save utf8 hash table
* Load utf8 hash table
* Find Needed Structures (Recurrent task)

## Define and Save Needed Structures - Save/Load heap

We have included parameters to run the JamVM in persistent mode. In this mode, the heap allocated memory is linked to a file. This file is used on subsequent persistent mode runs and is also used for debugging.

## Define and Save Needed Structures - Save utf8/classes hash table

We have created new functions to initialize hash tables in persistent mode. On these functions, the hash table allocated memory is linked to a file to be used on subsequent persistent mode runs.

## Define and Save Needed Structures - Find Needed Structures

After some investigation, we found out that chunk's headers have information that should be saved. The first two bits of a header are flags to check if the chunk has hash code and if the hash code is taken, respectively. The last 3 bits are also flags but further investigation is needed on those.

## Define and Save Needed Structures - Load utf8/classes hash table

Unfortunately, we weren't able to complete this task with the available time for this sprint.

## Organize Development Environment

* Install native Ubuntu
* Install Eclipse
* Install Git gui
* Import current sprint branch
* Boost VM performance

## Organize Development Environment - Install native Ubuntu/Eclipse

We have arrived to the conclusion that this is not an important or productive task. We will continue to use VMs because they have almost optimal performance and have some advantages (High replication and fast installation).

## Organize Development Environment - Install Git gui

We have installed Git gui on ours VM's.

## Organize Development Environment - Import current sprint branch

We have imported and we are all working on the same branch.

## Organize Development Environment - Boost VM performance

We have changed our VM's settings to use 4GB and 2 processors.

## Create Adapter Class

* Identify places in code where the team has made changes
* Create an "Adapter Class" to hold all changes
* Change code to use adapter class
* Adapt utf-8 hash table changes
* Adapt classes hash table changes
* Adapt heap initialization changes
* Search generic hash table manipulation functions
* Search in code calls for hash structures so we can adapt all relevant functions
* Create usage sample for UTHash hash tables

## Create Adapter Class - Identify places in code where the team has made changes

We have verified that the places where the team has made changes are:

* class.c - classes hash table manipulation
* alloc.c - heap initialization
* utf8.c - utf8 hash table manipulation

## Create Adapter Class - Create an "Adapter Class" to hold all changes

We have created an adapter class to fork the execution flow of the JamVM.

## Create Adapter Class - Change code to use adapter class

We have changed the code to utilize the adapted classes when running with our "persistentheap" parameter.

## Create Adapter Class - Adapt utf-8 hash table changes

After several different approaches, we have arrived on a problem where the JamVM tries to allocate a contiguous memory space and crashes due to memory allocation collision.

We have decided to reevaluate the adaptation and creation of new structures due to this problem.

## Create Adapter Class - Adapt classes hash table changes

We have arrived to the conclusion that here are no changes to be adapted yet, this task should be moved to next sprint.

## Create Adapter Class - Adapt heap initialization changes

We have decided not to apply an adapter to initialiseAlloc function because we were having trouble passing static variables on alloc.c to our initialiseAlloc\_adaptee.  
InitialiseAlloc is working fine and would have very few lines of code adapted so we understood that it was an unnecessary effort.

## Create Adapter Class - Search generic hash table manipulation functions

We have decided to use hash table manipulation library UTHash as it is open source and BSD licensed.

## Create Adapter Class - Search in code calls for hash structures so we can adapt all relevant functions

We have verified that hash structures calls that need to be adapted are:

* utf8.c:

-findHashedUtf8()

-initialiseUtf8()

* class.c:

-Depends on the object, functions are a little more delicate to adapt.

## Create Adapter Class - create usage sample for UTHash hash tables

We have Created two sample codes:

* usage-sample.c - Has examples for simple usage.

save-load-sample.c - Has example for saving hash table context into a file and loading into another hash table.

# Tests

We have created another initialization parameter called "testingmode".

We have created a test header that creates a log file. This header only runs if the testingmode flag is true.

# Support Software

## Github

We have been using github as our main repository for code. Our repository can be found in <https://github.com/icaro-henrique/JamVM-PH>.

## UTHash

Hash table manipulation library for C. Open Source, BSD licesended.

Found in <http://troydhanson.github.io/uthash/>.

## Eclipse

Java, C/C++ and PHP integrated development environment (IDE).

Found in <https://www.eclipse.org/>.

# Reviews

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| June/20/2014 | 1.0 | Sprint: Welcome to the Jam | Pedro Monteiro |
| July/17/2014 | 1.1 | Sprint: Pump Up the Jam | Pedro Monteiro |
|  |  |  |  |