Towards Next-Gen CRHM

Banani Roy, Manishankar Mondal, Hamid Khodabandehloo, Chanchal K. Roy, Kevin A. Schneider
{banani.roy, mshankar.mondal, hak335, chanchal.roy, kevin.schneider}@usask.ca
University of Saskatchewan, Canada

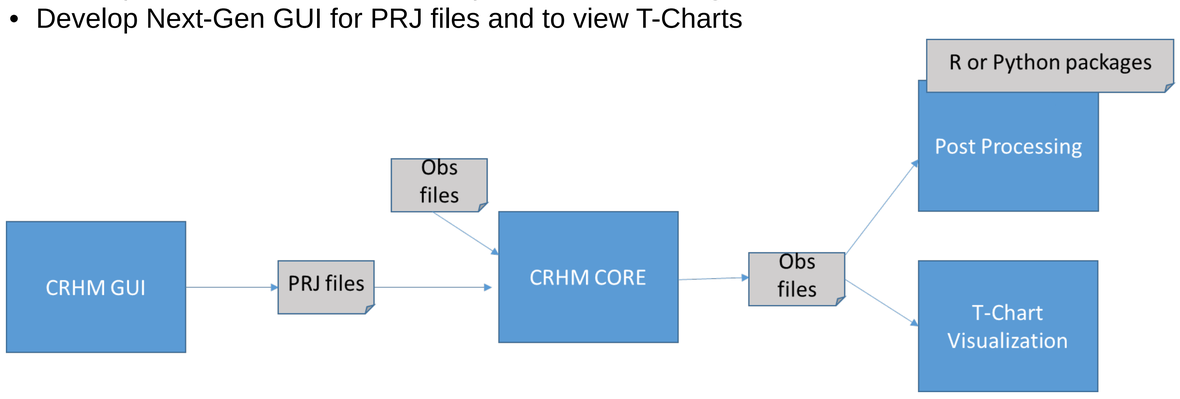
Global Water Futures
Solutions to Water Threats in an Era of Global Change

I. Abstract

The Core Computer Science Team has been working on migrating a GUI based legacy system called CRHM (Cold Region Hydrological Model) developed in Borland C++. CRHM has been widely used by worldwide Government Agencies(22), Universities (45), and Corporations (20). The system was initially devised to provide a framework within which to integrate numerical algorithms derived from the observation of a range of hydrological processes of considerable uncertainty, based solely on the underlying physical interactions which control them, in small to medium-sized catchments. CRHM's source code has 73.108 KLOC in 61 source files. Migration is essential for the system as the Borland C++ compiler is outdated and making modifications to CRHM is time consuming. We are working on producing the next generation CRHM.

II. Next – GEN CRHM

- Establish Next-Gen Development Environment
- Design new software architecture
- Create Plug-in Architecture
- Develop Workflow Framework with provenance tracking
- Develop Next-Gen GUI for PRJ files and to view T-Charts



III. Next-Gen CRHM Development Environment

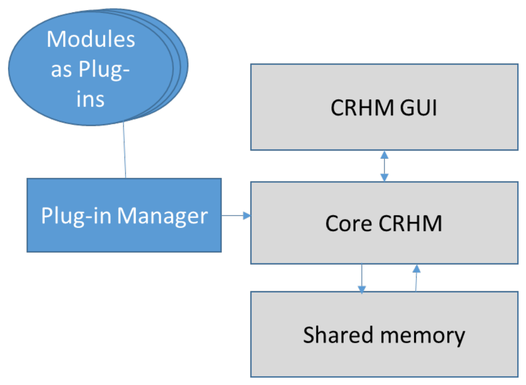
- A **modern compiler and libraries**
  - E.g., MS Visual C++ (VC++) 2017, GCC
- A Distributed **Version Control System** to support multi-version experimentation
  - E.g., Git
- An **Automated testing** framework to support rapid deployment of new features
  - E.g., test scripts, Google Test, Microsoft Coded UI, Microsoft TFS.
- **Virtual Containers** to support packaging, deployment & portability
  - E.g., Docker – Build, Ship and Run

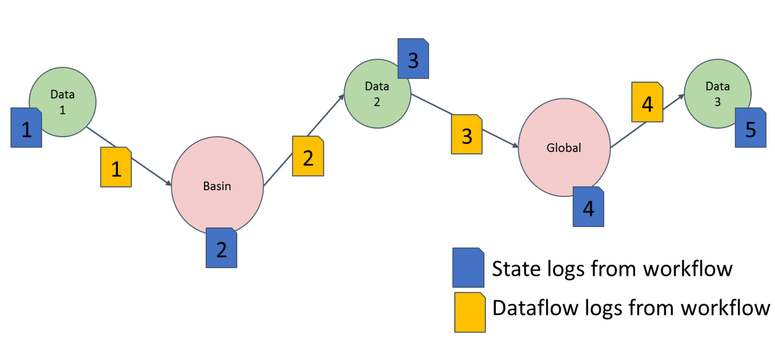
IV. Migration Strategies

- Code Separation: Core and GUI components.
- Develop a console version of CRHM 2018 that runs in a standard C++ environment.
- Create APIs to access CRHM core data structures.
- Use the API to populate components for GUI applications.
- Remove Borland dependencies.
- Minimize MFC dependency.

V. Next-Gen CRHM Architecture - Long Term

- Adding new modules as plugins
- Reducing inter module dependencies
- Connecting Modules in a workflow
- Establishing Workflow Provenance:
  - **Analyzing complex behavior** of workflows
  - **Identifying similarities** among workflows
  - **Finding the best parameter settings** for a module
  - **Resource management** and tracking





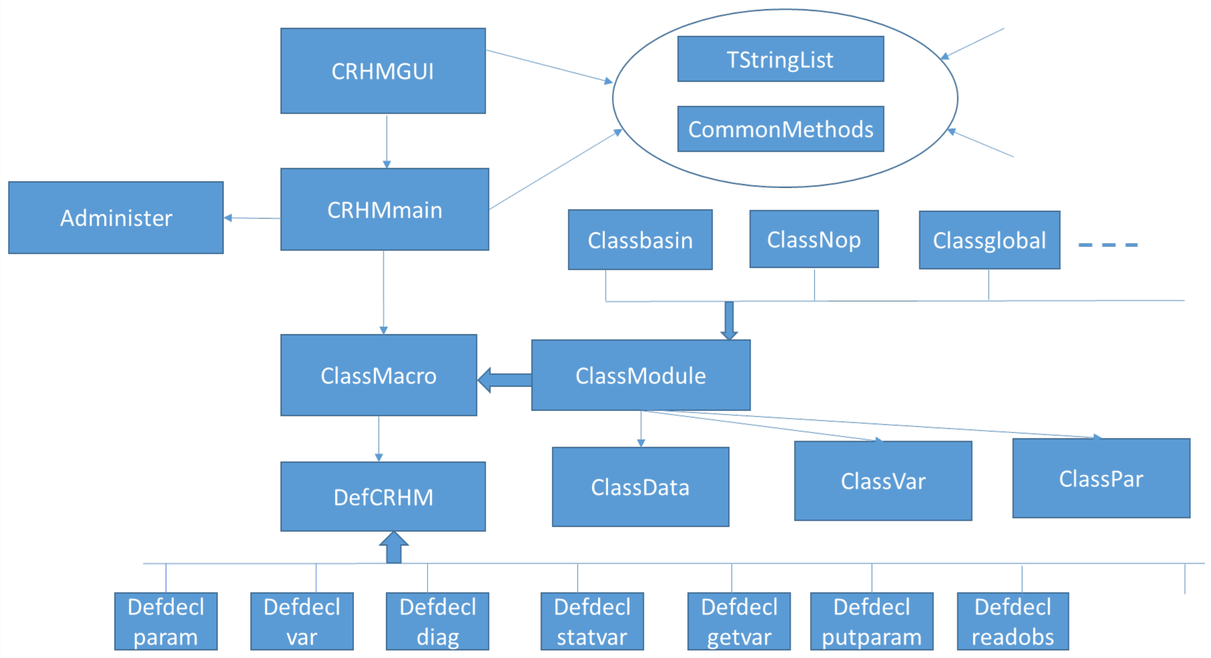
VI. Automated Testing

- We considered three major testing methods to make sure CRHM is working as expected.
- The first and an essential test for most software systems is **Unit testing**. We used **Google test** to develop robust and versatile unit tests.
- **System testing** is the second type of testing that we established for CRHM to ensure that its functionality as a whole is working properly. **Microsoft Coded UI** is used for this testing.
- Finally, we need to make sure that the results provided by the new application matches the legacy application. For this purpose we run test cases on both the old and the new applications automatically and compare the results. We consider this process as one part of **user acceptance testing**. To make sure the result is what an end user wants, one of the professional end users will confirm the final version before release.
- We use Microsoft TFS for maintaining test cases, bug reports and stories of the system.

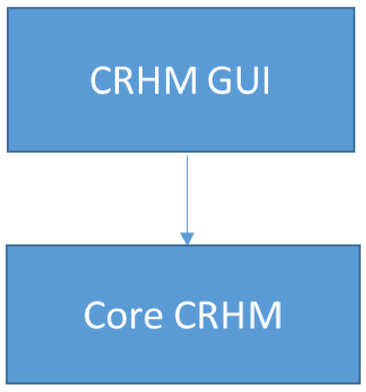
VII. CRHM Migration Accomplishments

- Analyzed different versions of CRHM
- Prototyped Migration of 2012 Version to modern compiler
  - Migrated CRHM 2012 Console version to Microsoft Visual C++ (MSVC++) 2017
- Prototyped CRHM GUI in MSVC++ 2017
- Created T-charts using observation files in MSVC++ 2017
- Partially migrated CRHM 2018 version to MSVC++ 2017 core version.
- Connected CRHM 2018 GUI MVSC++ prototype to partially migrated CRHM 2018 MSVC++ version
  - Open and run a project file
  - Integrated T-chart ActiveX control to show simulation results
  - Save and Save as functionality
  - Build Functionality
  - Close project file.
- Integrated migrated CRHM with Google Test Framework
- Setup a Github repository for version control

VIII. Next- GEN CRHM's Architecture – Current Status



IX. High-level Control Flow of Next-GEN CRHM




```
CRHMmain *crhm_core = CRHMmain::getInstance();

crhm_core_>DoPrjOpen(filename, ApplicationDir);

TStringList* variables = CRHMmain::getInstance()->getVariables();

for (int ii = 0; ii < variables->Count; ii++)
{
    std::string s = variables->Strings[ii];
    CString cvariables(s.c_str());
    listbox_variables_all.AddString(cvariables);
}
```

X. Observations

- CRHM code contains many exactly or nearly similar code fragments. These similar code fragments can be refactored for better maintenance and evolution of CRHM code. We are focusing on refactoring similar code during migration.
- There are many extra large functions in the CRHM code. These functions should ideally be decomposed considering the Single Responsibility Principle. We will do this as part of the migration.

XI. Conclusion and Future Work

The migration is on track according to the planned schedule. Both development and testing are being conducted side by side.

While migrating we are also considering the documentation for the functionality so that new developers and testers can easily contribute to CRHM in its maintenance phase. We are developing tools that will assist new CRHM developers in understanding CRHM's execution processes [1, 2].

In the future, we will use our CRHM migration experience for devising a semi-automatic mechanism for migrating and renovating similar software systems.

List of Publications

1. Banani Roy, Kevin Schneider, Chanchal Roy, Rayhan Ferdous and Mahjabin Alam. An Approach for effectively Understanding a Legacy Software System for Migration: A Case Study using a Borland Application. In progress.

2. Rayhan Ferdous, Banani Roy, Chanchal Roy and Kevin Schneider. ProvMod: A Workflow Programming Model to Offer Automated Provenance and Log Analysis. In *Proceedings of International Conference on Web Services (ICWS), 2019*. In progress.

Acknowledgement

We thank the CRHM developers for explaining CRHM's functionality. This work is supported in part by the Canada First Research Excellence Fund (CFREF) under Global Water Futures (GWF).