A Web-based Visual Analytic System for Understanding the Structure of Community Land Model

Yang Xu [1], Dali Wang [2 *], Tomislav Janjusic [3], Xiaofeng Xu [2]

[1] Department of Geography
University of Tennessee, Knoxville
TN 37996, USA
yxu30@utk.edu

[2] Climate Change Science Institute
Oak Ridge National Laboratory
TN 37831, USA
{wangd, xux4}@ornl.gov

[3] Computer Science and Mathematics Division
Oak Ridge National Laboratory
TN 37831, USA
janjusict@ornl.gov

* Corresponding author, Tel +1 8652418679, Fax +1 865 5749501. Email: wangd@ornl.gov

Abstract - *Development of high fidelity earth system models is important to the understanding of earth system science. Along with several decades of active developments, the complexity of the model's software structure became a barrier that hinders model interpretation and further improvements. In this paper, a web-based visual analytic system is introduced to better understand the software structure of Community Land Model (CLM) within an earth system modeling framework. First, the software structure is decomposed from source codes and we use a graph structure to represent the interrelationships among different CLM components. Second, a web-based front end is developed to demonstrate the CLM software structure in a visual analytical context. Finally, we present a pilot case study to discuss how an improved understanding of CLM software structure can be achieved from three different perspectives, namely CLM structure overview, visualization of submodel structure and CLM inter-version comparison. We believe the approaches and visualization tools can be beneficial to CLM model interpretation and improvements as well as other large-scale modeling systems across different research domains.*

Keywords - Community Earth System Model, Community Land Model, Software Structure Decomposition, Graph Visualization.

1 Introduction

Researchers have made great progress over the past decades in developing high fidelity earth system models [1]. The Community Earth System Model (CESM) is one of the leading earth system models funded by National Science Foundation (NSF) and U.S. Department of Energy (DOE). The Community Land Model (CLM) is the land model of CESM that simulates surface energy, water, carbon, and nitrogen fluxes and state variables for the land surfaces [2-4]. The model formalizes and quantifies concepts of ecological climatology under an interdisciplinary framework to understand how natural and human changes in vegetation affect climate. As a scientific application for the earth system simulation, it is important to get the fundamental processes correct [5]. This requires a good understanding of CLM ecosystem functions as well as the interplay among them within the context of ecosystem science.

The CLM contains several submodels related to land biogeophysics, biogeochemistry, hydrologic cycle, human dimensions and ecosystem dynamics. The structure of each submodel is generally organized by software modules or subroutines based on natural system functions such as carbon-nitrogen cycles, soil temperature, hydrology and photosynthesis [6]. Each module or subroutine interacts with a list of variables which are globally accessible or subroutine explicit. Several efforts have been made to better understand CLM and the ecosystem processes through software structure profiling [6], functional unit testing [7] and memory pattern analysis [8]. The whole CLM modeling system consists of more than 1800 source files and over 350,000 lines of source code. New CLM software analysis methods are much needed for rapid model interpretation and improvements.

In this paper, a web-based visual analytic system is introduced to gain an improved understanding of CLM software structure. First, we decompose the CLM software from source codes and propose a CLM graph structure that summarizes the interrelationships among all the function calls and variables. Second, a web-based front end with three different views is developed to demonstrate the CLM software structure in a visual analytical context. A pilot case study is then presented to gain insights into the structure using the three views, namely CLM structure overview, visualization of submodel structure, and CLM inter-version comparison. We believe the visualization tools can be beneficial to the understanding of CLM software structure. The approaches can also be applied in other large-scale modeling systems across different research domains.

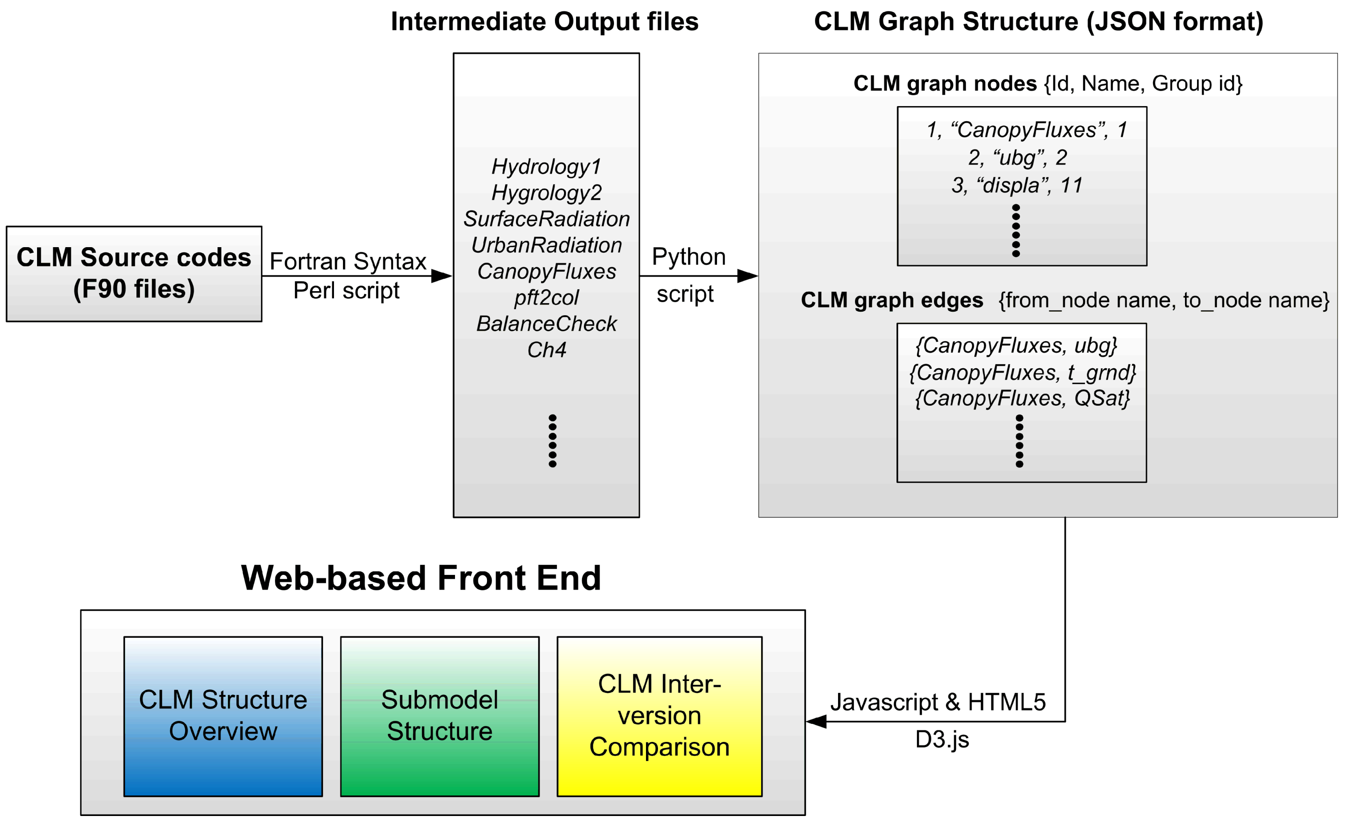

Fig. 1. Work flow and key components of the web-based visual analytic system for understanding the structure of Community Land Model (CLM)

2 Methodology and key components

In this section, we introduce the key components and work flow of our web-based visual analytic system. As shown in Fig. 1, a CLM Fortran-syntax specific Perl script was developed to decompose the CLM software structure into tokens of function calls, subroutine explicit parameters and global variables. Definition of tokens will be further explained in section 2.1. Then, a Python script builds a CLM graph structure, which summarizes the interrelationships among all the tokens. Finally, the graph structure is visualized in the web-based front end using Javascript and D3.js (http://d3js.org).

2.1 Decomposition of CLM structure

Understanding complex codes such as CLM undoubtedly requires tools to facilitate code decomposition into simpler forms. This allows users to use visualization tools to further understand the code structure. For this purpose, we developed a CLM Fortran-syntax specific Perl script that categorizes key variables and data structures into tokens. Herein, we refer to a token as any source-code identified function call or variable, which includes name of subroutines, global visible variables, as well as all the variables used in subroutine definitions *(subroutine-in, subroutine out)*. *Subroutine-in* variables are all tokens identified in the subroutine's signature. *Subroutine-out* variables are a subset that was identified to be written to, i.e. these tokens were used to store a value. Globally visible variables are identified using the pointer assignment syntax during source-code scanning. This means that any token found in the source-code line that adheres to the general pointer assignment syntax is treated as globally visible variable. We further break this category into *Read-only, Write-only*, or *Modified* variables. Specifically, during scanning the script stores any pointer to derived member values into a hash of tokens. The source code lines are decomposed into left-hand (lHand) and right-hand (rHand) statements and further broken down if more assignments are present. Every token found on the lHand side is a write category and similarly every token on the rHand side is a read category token. If a token falls into both categories, we will assign that token into the modified category. There are, of course, special cases that require further statement breakdown using special-case rules. For example, statements that use pointers to access other derived types are often found in the lHand/rHand statement syntax, thus the script will decompose tokens to identify the correct category for the globally visible variable. In addition, tokens found in source-code statements that are identified by special-keywords (e.g., call) are used to build static call-graphs, which is a list of calle subroutine names originating from the current caller subroutine. The Perl scanning process outputs a list of files named after the subroutine's name. Each file records the variables and function calls (Calle Subroutines) that a particular subroutine has accessed. Table 1 gives an example of the output file of *CanopyFluxes* subroutine.

Table 1. Tokens of Cannopyfluxes Subroutine

Category	Tokens
Subroutine-In	*ubg, ubc, lbg, lbp, num_nolakep,* ......
Subroutine-Out	*Null*
Global Read Only	*t_grnd, psnsun_wc, alphapsnsun, psnsun,* ......
Global Write Only	*cgrnd, psnsun, rb1, ulrad, dlrad,* ......
Global Modified	*displa, rc13_psnsun, z0qv, z0hv,* ......
Global None	*watopt, watdry*
Function Calls	*QSat, FrictionVelocity, Photosynthesis,* ......

2.2 Graph construction of CLM software and submodels

Based on the output files generated by the Perl script, we developed a Python script to organize the CLM components into a graph structure with nodes and edges. The nodes refer to all the identified tokens, and the edges are used to describe how these tokens access or are accessed by others. We use this graph structure to summarize the interrelationships among all the function calls, subroutine explicit parameters and global variables. As described in section 1, CLM consists of several submodels and each submodel is usually organized by particular subroutines. In order to incorporate this information into the graph, the Python script also records which submodel each subroutine belongs to. Then this graph structure is used in the web-based front end to facilitate the understanding of the CLM structure from multiple perspectives. For example, users could get an overall idea of CLM structure by exploring the submodels and subroutines contained, or look into the structure of particular CLM submodels. The graph structure can also be compared across different CLM versions.

Due to the complexity of CLM software, some tokens (nodes) belong to multiple categories in the modeling context. For example, in Table 1, the token *displa* marked as *Global Modified* variable for the subroutine *CanopyFluxes* while its category becomes *Subroutine-in* for another subroutine *FrictionVelocity*. In order to maintain this information, we generate a CLM node group list that enumerates all possible combinations of the token categories. As shown in Table 2, each Group id corresponds to a particular combination of token categories. By introducing the list, we are able to label each node with group information during graph construction. For example, as shown in Table 1, all *Function Calls* (e.g., *Qsat*) after graph construction will have a Group id of 1. The *Subroutine-in* variables (e.g., *ubg*) will have a Group id of 2. While variables like *displa* as described above will have a Group id of 11. The group information can be used to understand the CLM software structure with respect to token category, i.e. its function.

Table 2. CLM Node Group Information

Group id	Combination of Token Categories
1	*Function Calls*
2	*Subroutine-in*
3	*Subroutine-out*
4	*Global Read-only*
5	*Global Write-only*
6	*Global Modified*
7	*Global-None*
8	*Subroutine-in & Subroutine-out*
9	*Subroutine-in & Global Read-only*
10	*Subroutine-in & Global Write-only*
11	*Subroutine-in & Global Modified*
12	*Subroutine-in & Global-None*
⋮	⋮

2.3 Web-based front end based on Javascript and D3.js

One of the important components in our system is the web-based front end, which is designed to facilitate the exploration and investigation of CLM software structure in a visual analytical context. The web-based front end has three major views: CLM structure overview, visualization of submodel structure and CLM inter-version comparison. The CLM structure overview aims to provide users with an overall picture about different CLM software versions and submodels. The visualization of submodel structure summarizes the inter-relationships among all the function calls, subroutine explicit parameters and global variables related to that submodel (e.g., *CanopyFluxes* shown in the next section). The inter-version comparison is used to demonstrate what changes and improvements have been made from one CLM software version to another. The web-based front end is developed based on Javascript and D3.js (http://d3js.org). D3.js is a Javascript library which allows developers to bind their data to a Document Object Model (DOM) and then transfer the data information into interactive visualizations. As shown in Fig. 1, the Python scanning process generates a list of node and edge files in JSON format (http://json.org). These files that record the CLM software structures are used to create interactive visualizations using Javascript and D3.js.

3 A pilot case study

As we mentioned before, visualizing and analyzing the software structure of large-scale modeling system such as CLM is very important to model interpretation and provides opportunities for further improvements of model structures. In this section, a pilot case study is introduced to describe how an improved understanding of CLM software structure can be achieved with the web-based visual analytic system. First, a collapsible tree is used to demonstrate the overall structure of CLM software from a hierarchical perspective. This effort will allow users to explore the submodels and subroutines included in each CLM version. Second, a directed graph is used to visualize the *CanopyFluxes* submodel within CLM. This effort

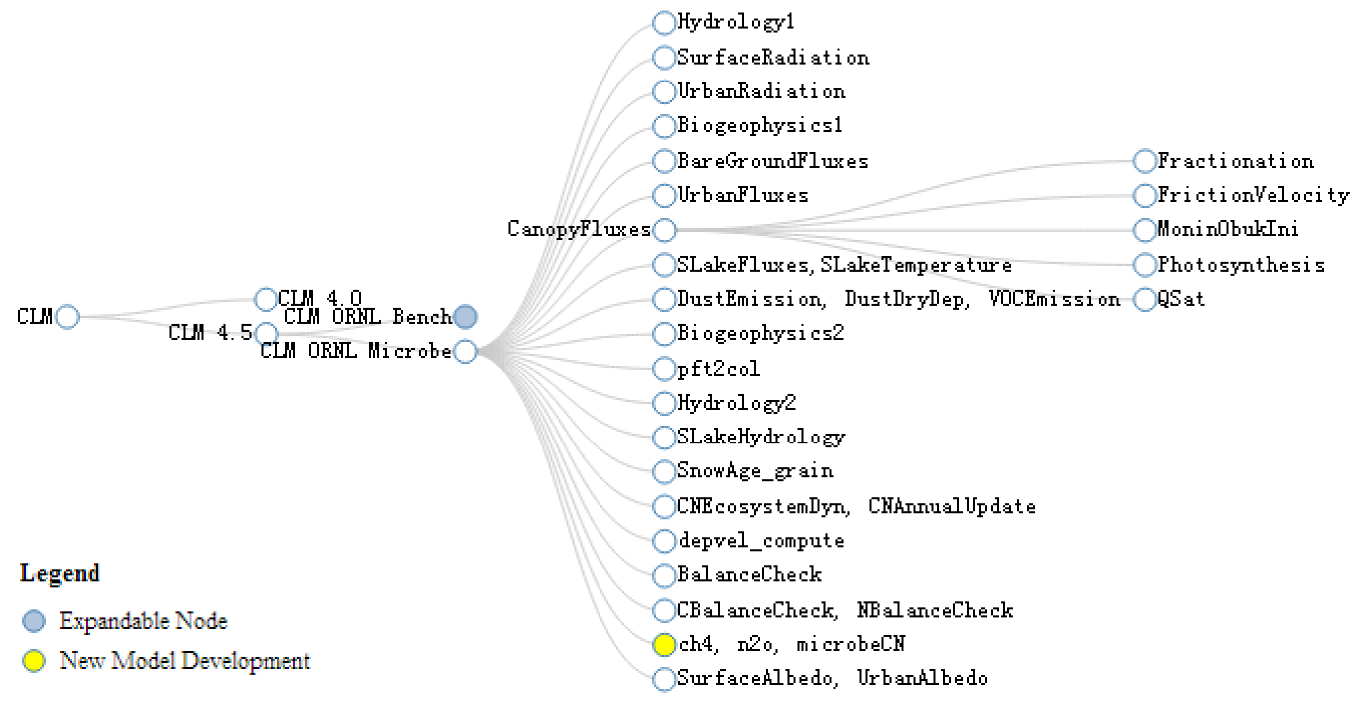

Fig. 2. Visualization of CLM software structure using a hierarchical collapsible tree. The nodes can be clicked to expand or collapse. The nodes in light blue denote the ones that can be expanded. The nodes in yellow represent new model development (e.g., ch4, n2o, microbeCN) based on the bench version (e..g, CLM ORNL Bench).

allows users to further look into the sturcture of particular submodels based on their research interests. Finally, we present our case study for the CLM inter-version comparison (*CLM ORNL Bench* vs. *CLM ORNL Microbe*). This effort enables users to trace the changes between two CLM versions.

3.1 CLM structure overview

The CLM model has several public releases such as CLM 4.0 and CLM 4.5. At Oak Ridge National Laboratory (ORNL), we have our own code repository, which use the official release CLM 4.5 as our bench case. Based on that, several new modules (e.g., the Microbe module [9]) have been developed. In our web-based visual analytic system, a collapsible tree is used to demonstrate the CLM software structure from a hierarchical perspective. As mentioned in section 2.1, after the decomposition of CLM structure, we generate a list of files named after the subroutine's name, which can be used to construct the overview software structure. Fig. 2 shows a hierarchical tree, which allows users to explore the CLM software structure by expanding or collapsing particular nodes. For example, when users click the node "*CLM*", it will expand and show several CLM major release such as "*CLM 4.0*" and "*CLM 4.5*". The node "*CLM 4.5*" can be then expanded to view different versions of CLM source codes such as "*CLM ORNL Bench*" and "*CLM ORNL Microbe*". Each CLM2 version can be further expanded to view its submodels (e.g., "*CanopyFluxes*") as well as the corresponding subroutines. The visualization also highlights the nodes in yellow to illustrate the submodels which are newly developed based on the CLM bench version. For example, the node "*ch4, n2o, microbeCN*" in Fig. 2 shows that this submodel is newly incorporated in "*CLM ORNL Microbe*" as compared with its bench version "*CLM ORNL Bench*".

The hierarchical visualization provides users with an overall picture of CLM software release, submodel components as well as the model improvements. The interactive visualization can be found at (http://web.ornl.gov/~7xw/CLM_Overview.html).

3.2 Visualization of submodel structure

Visualizing the structure of CLM submodels can be useful when users want to understand particular CLM components at the micro level. As described in section 2.2, the Python script generates a comprehensive graph structure recording all the submodels and their corresponding subroutines. Within each subroutine, the interrelationships among all the function calls and variables are recorded. Hence, the web-based front end is able to visualize a subset of the whole graph in order to demonstrate the structure of a particular CLM submodel.

Fig. 3 shows the structure of *CanopyFluxes* submodel within CLM. Nodes with different colors and sizes are used to denote the types of the tokens. Nodes with bigger size and the color of yellow stand for all the function calls (subroutines) for the *CanopyFluxes* submodel. The nodes with smaller size denote all the variables and among these variables: (1) green nodes stand for subroutine explicit parameters; (2) blue nodes stand for global variables; (3) the nodes in red denote the ones that are used as both subroutine explicit and global variables.

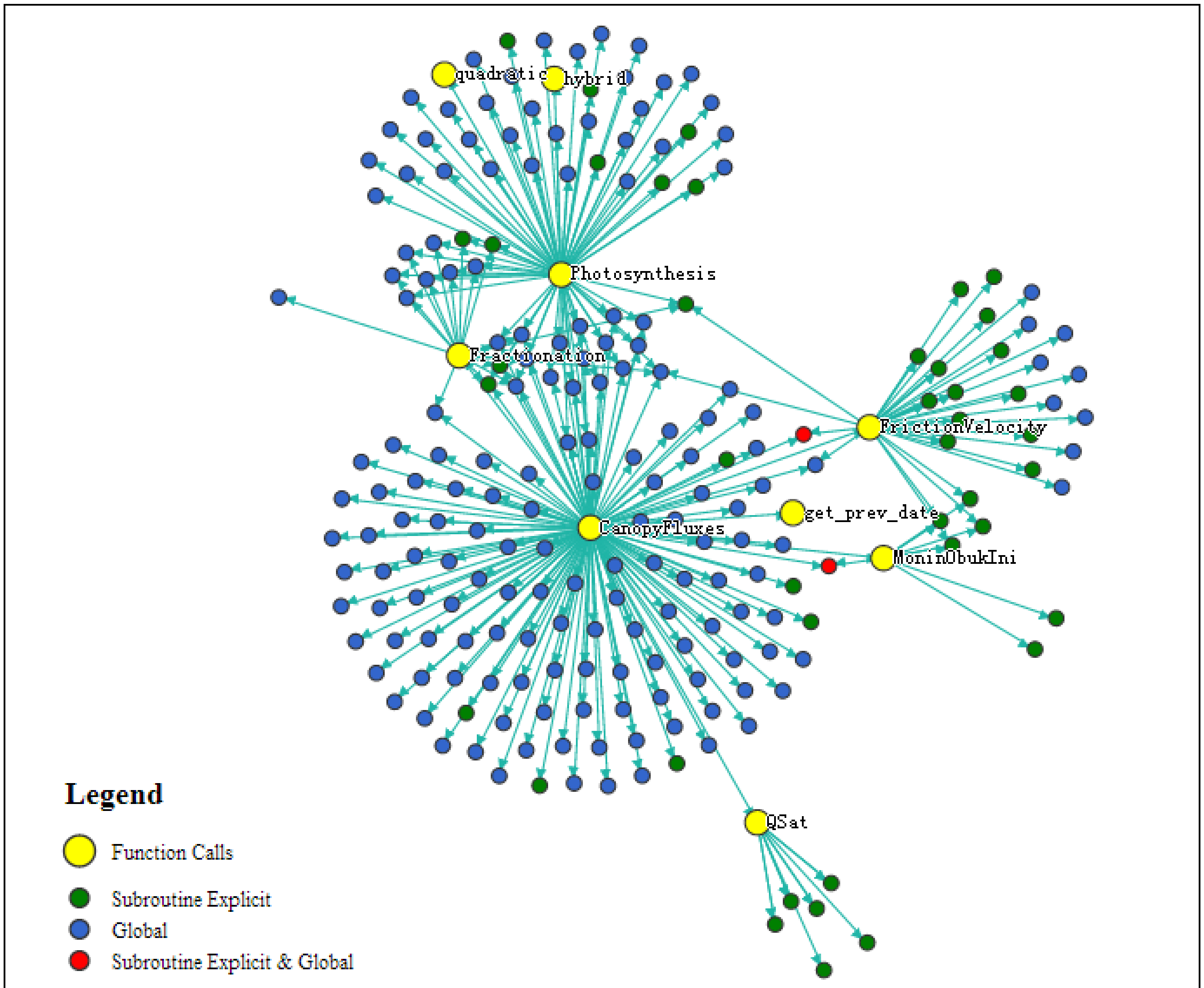

Fig. 3. Visualization of *CanopyFluxes* submodel of CLM ORNL Microbe. Nodes in yellow stand for the function calls (subroutines). Nodes in green denote subroutine explicit variables. Nodes in blue represent all the global variables. While nodes in red denote the variables which are used as both subroutine explicit/global variables. The edges are used to describe how the variables are accessed by particular function call(s).

The visualization using directed graphs provide users with an intuitive way of investigating how function calls and variables access or are accessed by others within the context of particular submodel. By exploring the submodel structure, users could better understand how the tokens are connected together as well as the specific role(s) that each of them is playing. As shown in Fig. 3, several red nodes exist in the structure of *CanopyFluxes*, which means that these variables serve as both global variables for the submodel and explicit parameters for the function calls (subroutines). The information allows users to further explore the scientific meanings of these variables.

The case study of this visualization can be found at (http://web.ornl.gov/~7xw/CanopyFluxes/CanopyFluxes.html) . When a user puts the mouse over a certain node, the name as well as the node group id will pop up. The group id, as described in Table 2, offers users detailed information about the specific category of an token. For example, the pop up information will help users to distinguish if a global variable belongs to the category of *Read-only, Write-only* or *Modified*. Our web page contains a hyperlink (i.e., "View Group Information") which leads to a file similar to Table 2 whenever users want to view the group information of the tokens.

3.3 CLM inter-version comparison

The CLM is a community model which is open for any contributions and usages across the scientific community. For example, the current release of CLM 4.5 (i.e., *CLM ORNL Bench* in our case study) consists of four key components: biogeophysics, hydrologic cycle, biogeochemistry and dynamic vegetation. For the biogeochemistry component in the CLM, the carbon and nitrogen cycling in the soil and vegetation under the influence of environmental factors are simulated. The microbial controls on carbon and nitrogen processes are implicitly represented as a few empirical equations, which are one of the primary uncertainties for model improvements targeting better predicting biogeochemistry-climate feedbacks. Thus, a more advanced model with explicit representation of microbial processes which contributes to the soil biogeochemical processes is needed. Targeting this need, a new modeling structure, *CLM ORNL Microbe*, is developed to improve the *CLM Bench* version by [9, 10].

Our web-based front end uses a rendered directed graph to describe the changes between two CLM versions. As described in section 2, the CLM graph structure (nodes and edges) that summarizes the interrelationships among all the tokens is generated using Python script. By comparing the graph structures between two different versions, we are able to uncover the changes in term of: (1) which function calls and variables are newly added, modified or no longer existing; (2) the interplay between function calls and variables that are newly added or no long existing. Fig. 4 shows an example of our web-based visualization for comparing *CLM ORNL Bench* and *CLM ORNL Microbe*. The *CLM ORNL Bench* is chosen as the base version. The blue nodes denote the newly added function calls and the yellow nodes stand for the function calls that are modified. For example, we can see several new function calls that are added into the *CLM ORNL Microbe* such as *n2o*, *microbeCN* and *microbeRest*.

The nodes with smaller size denote all the variables. We use red to represent newly added variables and green to represent the ones that are modified. We also use solid and dash lines to represent the changing relationships among all the tokens through the two CLM software versions. For example, as shown in Fig. 4, there are many solid as well as dashed links associated with the function call *ch4*. It means that as compared with the based version, the *ch4* submodel for *CLM ORNL Microbe* is modified to access some new variables (red nodes connected with solid lines). Meanwhile, some of the variables (green nodes) and function calls (yellow nodes) are connected

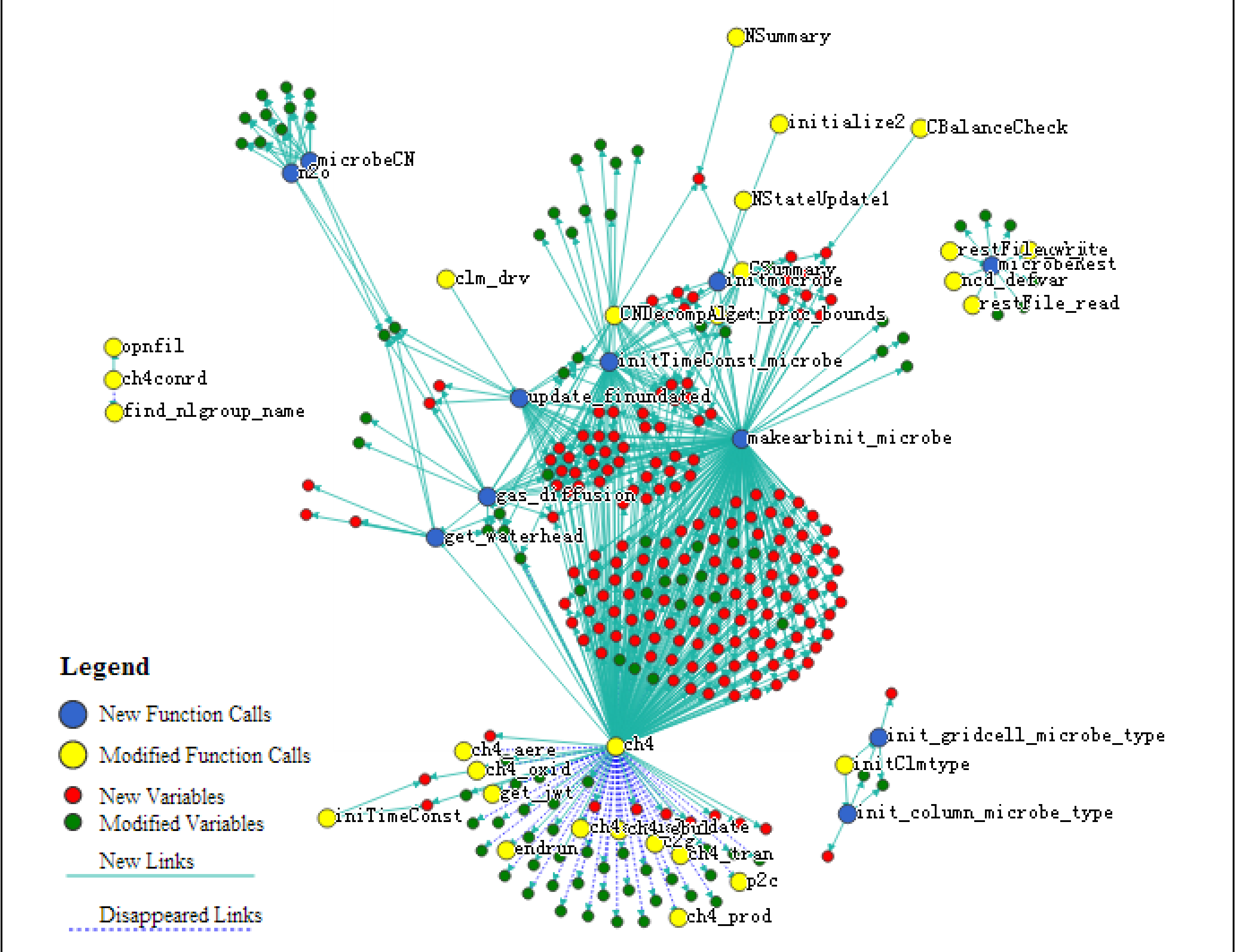

Fig. 4. Graph visualization of CLM inter-version comparison (*CLM ORNL Bench* vs. *CLM ORNL Microbe*). The *CLM ORNL Bench* is chosen as the base version. Blue nodes represent newly added function calls and the yellow nodes represent the modified function calls. Red nodes denote the newly added variables and the onw in green stand for modified variables. The links rendered with solid and dash lines are used to represent newly added and disappeared links respectively.

to *ch4* with dash lines, which means that those variables and function calls are accessed by *ch4* in the *CLM ORNL Bench*, but are no longer accessed by *ch4* in *CLM ORNL Microbe*. The graph structure allows users to easily trace the changes between two CLM software versions in a visual analytical context. The web-based visualization for this case study can be found at (http://web.ornl.gov/~7xw/CLM_Microbe_Comparison/CLM_Comparison.html).

4 Conclusions and future work

In this paper, we present our approaches for better understanding the structure of Community Land Model within the Earth System Modeling framework. A web-based visual analytic system is developed to allow users to gain insights into software and data structure from different perspectives. The CLM structure overview provides an overall picture of different CLM release and the submodels. It helps users to explore major components as well as new module development for a particular CLM version (e.g., *CLM ORNL Microbe*). The system also enables users to look into the structure of a particular submodel by visualizing the interrelationships among all the function calls and variables. Moreover, a deeper understanding can be obtained by exploring their names and categories, which is important for model interpretation and further improvements. The CLM inter-version comparison allows users to trace the changes between two CLM versions in a visual analytical context. Users can easily identify the function calls and variables which are added, modified or removed from one CLM version to the other. We believe the approaches and visualization tools can be beneficial to the understanding of CLM software structure as well as other large-scale modeling systems across different research domains.

The future work will focus on two directions. First we will develop an online database system hosting software structure as well as performance data from other advanced tools such as Vampir [11] and Valgrind (www.valgrind.org). Users will be able to query the database to get detailed information of CLM submodels, function calls and variables. Meanwhile, visualizations of software and submodel structures will be generated on the fly based on the customized queries. Second, we will incorporate a web-based functional testing platform over the cloud computing infrastructure to facilitate the understanding of CLM ecosystem processes.

Acknowledgment

This research was partially funded by the U.S. Department of Energy (DOE), Office of Science, Biological and Environmental Research (BER). This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. Yang Xu's work is partially supported through a subcontract to the University of Tennessee, Knoxville. Oak Ridge National Laboratory is managed by UT-Battelle LLC for the Department of Energy under contract DE-AC05-00OR22725.

References

[1] W.M. Washington and C.L. Parkinson, *An Introduction to Three-Dimensional Climate Modeling*, 2nd ed. University Science Books, 2005.

[2] G.B. Bonan, "The Land Surface Climatology of the NCAR Land Surface Model Coupled to the NCAR Community Climate Model", *J. of Climate*. vol. 11(6), pp.1307-1326, 1998.

[3] R.E. Dickinson, K.W. Oleson, G. Bonan, F. Hoffman, P. Thornton, M. Vertenstein, et al. The Community Land Model and Its Climate Statistics as a Component of the Community Climate System Model, *J. of Climate*. Vol. 19(11), pp.2302-2324, 2006.

[4] K. Oleson, D. Lawrence, B. Gordon, M. Flanner, E. Kluzek, J. Peter , et al. "Technical Description of Version 4.0 of the Community Land Model (CLM)", 2010.

[5] D. Wang, D. Ricciuto, W. Post and M. Berry. "Terrestrial Ecosystem Carbon Modeling", *Encyclopedia for parallel Computing*, pp.2034-2039, 2011.

[6] D. Wang, J. Schuchart, T. Janjusic, F. Winkler and Y. Xu, Toward Better Understanding of the Community Land Model within the Earth System Modeling Framework. *International Conference on Computational Science*. 2014. In press.

[7] D. Wang, Y. Xu, P. Thornton, A. King, C. Steed, L. Gu and J. Schuchart. "A Functional Test Platform for the Community Land Model", Environment Modeling & Software, vol. 55, pp. 25-31, 2014.

[8] T. Janjusic, K. Kavi and B. Potter. "A Memory Analysis Tool", *Procedia Computer Science*. Vol. 4, pp. 2058-2067. 2011.

[9] X. Xu, J.P. Schimel, P. Thornton, X. Song, F. Yuan, S. Goswami. "Substrate and Environmental Controls on Microbial Assimilation of Soil Organic Carbon: A Framework for Earth System Models". *Ecology Letters*, DOI: 10.1111/ele.12254. 2014.

[10] X. Xu, D.A. Elias, D.E. Graham, T.J. Phelps, S.L. Carrol and P. Thornton. "A Microbial Functional Group Based Model for Simulating CO_2 and CH_4 Dynamics". unpublished.

[11] M.S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, et al. "Developing Scalabe Applications with Vampir, VampirServer and VampirTrace". *Parallel Computing: Architectures, Algorithms and Applications*. Vol. 15, pp. 637–644. 2008