STView: An Eclipse Plug-in Tool for Visualizing Program Structures in Fortran Source Codes

Tomomi Ohichi[1], Masaaki Terai[2], Mitsuo Yokokawa[1,2] and Kazuo Minami[2]
1) Kobe University, 2) RIKEN Advanced Institute for Computational Science (AICS)

Background and Motivation

- To achieve more realistic simulations, modern applications have become more complicated , have more source modules than those previously used.
- Engineers need to understand program stuctures of an application before they modify source codes to improve its perfomances. However, the task of reading codes requires much effort and time.
- There are not many tools satisfied our requirements as follows.
  - visualization of program structures including loops and branches
  - a non-proprietary tool under the open-source software license
  - parsing source codes of an entire application in Fortran 90 or higher

→ STView

We developed an Eclipse plug-in tool using Photran [1].

Software Requirements:
- Photran ver. 9.0.1 or higher
- Eclipse ver. 4.4.1 or higher
- Java Runtime Environment ver. 1.8

Overview of STView

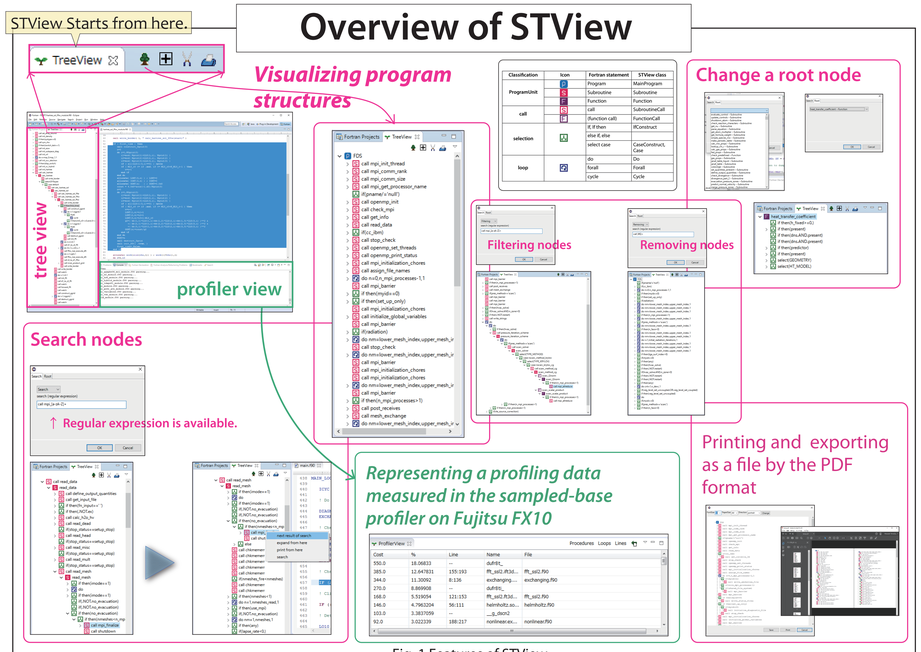


Fig. 1 Features of STView

Design and Implementation

STView performs the following three steps to visualize a tree.

STEP-1
- STView obtains opened files in the editor of Eclipse, collect file names of Fortran source code in folders including the opened files, and create a list of the filenames.

STEP-2
- Photran's parser generates ASTs for each program unit (e.g., program, subroutine and function) based on the filename list as shown in Fig. 2.
- STView creates a new STView-AST that only consits of loop, branches and procedure calls to reduce the number of nodes of the tree using the algorithm as shown in Fig. 3.
- STView ignores dependencies among program units, does not use a building script (e.g., Makefile), and creates a single tree of the entire program based on node identifiers as shown in Fig. 4.
- When STView detects a recursion structure that a callar procedure calls itself or its ancestor more than once, it eliminates the repeated callee procedures.

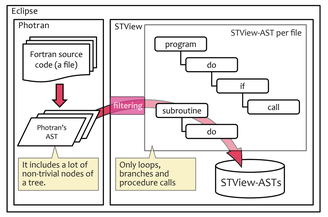


Fig. 2 Overview of transformation from Photran's AST to STView-AST based on the filename list

```
/* This algorithm creates STView-ASTs (treeList) from
   Photran's AST by removing unnecessary information.   */
input : Photran'sAST: an AST generated by Photran's parser
output: treeList: a set of STView-ASTs. Each member is a tree
        represents the structure of a program unit.
1  Algorithm algorithm()
2    treeList ⟵ ∅
3    foreach pnode ∈ Photran's AST do
4      construct(pnode, null)
5    end foreach
6  end
7  Function construct(pnode, snode)
8    if pnode is a program unit node then
9      cnode ⟵ create an STView-AST node correspoding to pnode
10     add cnode to treeList
11     snode ⟵ cnode
12   else if pnode is a loop/branch/procedure-call node then
13     cnode ⟵ create an STView-AST node corresponding to pnode
14     add cnode to treeList (as a child of snode if snode != null)
15     snode ⟵ cnode
16   end if
17   foreach c ∈ pnode's children do
18     construct(c, snode)
19   end foreach
20 end
```

Fig. 3 Algorithm of the tree construction
(A part of the STView-AST construction method)

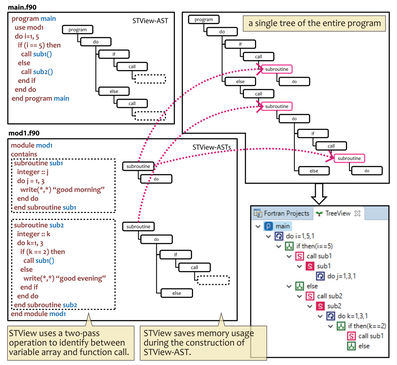


Fig. 4 Construction of a tree from STView-ASTs

STEP-3
- STView visualizes a tree based on the STView-AST.

Evaluation

- STView created visual program structures for all applications (Table 1) without any critical errors as shown in Fig. 5.
- Even though the current version of STView did not support the interface statement in Fortran, STView ignores a part of the source code and approximately visualizes program structures in the applications: "AxiSEM", "cftb", "FDS", "gtc", "MODYLAS", and "MOPAC."
- We confirmed STView is useful to know the program structures and shows much promise for practical use.

Fig. 5 Examples of visualizing program structures

Table 1 Sample Applications for the evaluation of STView

Name	Description	File ext.	#files	SLOC
AxiSEM-1.1 [6]	Spectral element method	F90+f90	80	52690
calypso [7]	Magnetohydrodynamics	F90+f90	694	17498
cftb [8]	Crystal field tight binding	f+f90	189	55244
ever [9]	Cosmological evolution simul.	f90	23	5504
FDS-6.5.1 [10]	fire-driven CFD (LES)	f90	35	144268
gtc [11]	Gyrokinetic toroidal simul.	F90	32	12809
MODYLAS-1.0.2 [12]	Molecular dynamics	f	41	40399
MOPAC-7.1 [13]	Quantum chem. (MO)	F90+f90	590	64132
NICAM-DC-1.1 [14]	Global climate simul.	f90	130	55300
RSDFT-1.2.2 [15]	Real-space DFT	f+f90	212	62656
SHF_OMP [16]	Pseudospectral method	f90	30	37491
SMASH-1.1 [17]	Quantum chem.	F90	33	75934
ver [18]	Radiative tranfer eqn. solver	f90	34	1585

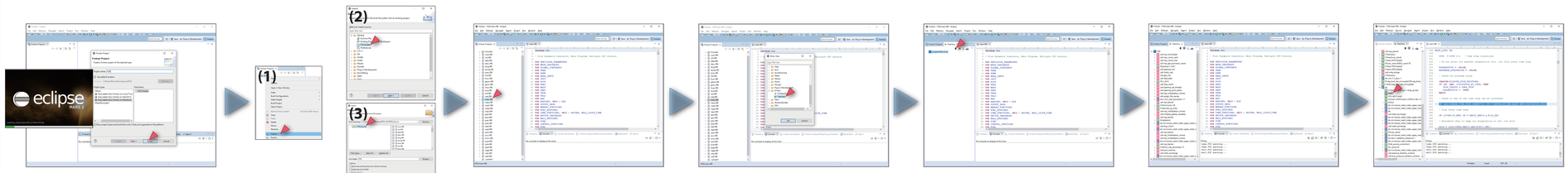
Future work

- In the current version of STView, the following features are unavailable. To be made even more useful, we'll implement them in the future.
  - Interface statement in Fortran
  - Preprocessing to directives (e.g., #if, #ifdef)
- Finally, we'll publish STView online as an open source.

Related work

- **FTNCHECK** [2] and **Source Navigator** [3]
  - Refactoring tools
  - Only call-tree without loops and branches
  - Fortran 77 or limited features of Fortran 90
- **K-scope** [4]
  - Fortran 77/90 source code analysis tool
  - Visualizing program structures including loops and branches
  - K-scope uses the external parser [5] that is underway to develop.

Visualizing program structures on Eclipse



(a) Start Eclipse and create a new Fortran project

(b) Import source codes to the created project

(c) Open a file on the editor

(d) Show the tree view from the menu bar

(e) Click the icon on the tree view, and then start a tree construction

(f) Completion of the construction, drawing a tree on the tree view

(g) Each node of the tree links correspondent a part of source code (e.g., subroutine, function and line of code) and show them on the editor

Fig. 6 Visualizing program structures on Eclipse

References

[1] Photran, http://www.eclipse.org/photran/.
[2] FTNCHECK. http://www.dsm.fordham.edu/ftnchek/.
[3] Source Navigator. http://sourcenav.sourceforge.net/.
[4] M. Terai at el. Extending K-scope fortran source code analyzer with visualization of performance profiling data and remote parsing of source code. In 2014 IEEE Inter. Conf. on High Perf. Comp. and Comm., pages 866–873, 2014.
[5] OmniXMP compiler, http://comni-compiler.org/.
[6] AxiSEM, http://seis.earth.ox.ac.uk/axisem/.
[7] calypso, https://github.com/geodynamics/calypso/.
[8] cftb, https://github.com/alexurba/cftb/.
[9] evora, https://github.com/jochenklar/evora/.
[10] FDS, https://github.com/firemodels/fds-smv/.
[11] gtc, https://github.com/shmilee/gtc/.
[12] MODYLAS, http://www.modylas.org/.
[13] MOPAC, http://openmopac.net/.
[14] NICAM-DC, http://scale.aics.riken.jp/nicamdc/.
[15] RSDFT, https://github.com/j-iwata/RSDFT/.
[16] SHF_OMP, https://github.com/rocketdude/SHF_OMP/.
[17] SMASH, http://smash-qc.sourceforge.net/.
[18] vef, https://github.com/bcfriesen/vef/.

Acknowledgement

Part of the work was supported by JSPS KAKENHI Grant Number JP16H02822.