SPINACH

Swarmed Parallel Interpreter for Numerical Analysis with Collaborating Hubs

Detailed A-Level Specifications
Version 1.1

Yehong Wang
10/19/2009

1. Overview

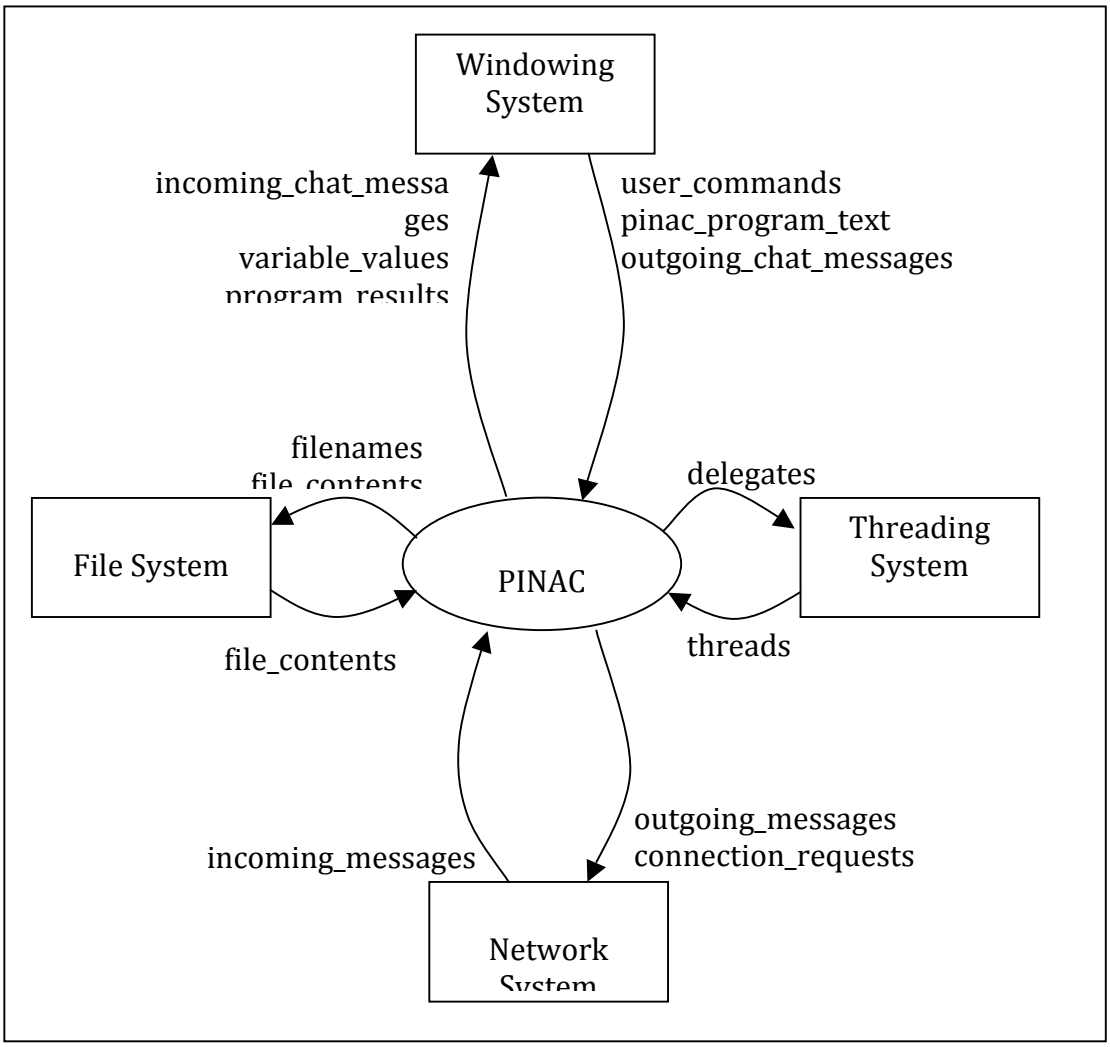

Figure 1 - Top Level Context Diagram

2. Referenced Documents

1. final_project_introduction.ppt – September 24, 2009
2. final_project_statement_of_work.doc – September 24, 2009
3. final_project_requirement.doc – September 24, 2009
4. SPINACH - Architectural Concept Document ver 1.1.docx – October 19, 2009

3. Requirement

3.1. Function Requirement

3.1.1. Interpreter Front End Team

3.1.1.1. **Shall** support interpreting SPINACH input syntax including:

a. creation of variables (strings and scalars or matrices of either doubles or integers);

b. creation of structures;

c. matrix operations (multiplication, addition, subtraction, transpose); scalar operations (multiplication, addition, subtraction, dot product)

d. assignment;

e. support for non recursive functions that can accept multiple values and return single value;

f. if statements;

g. tests for equality and inequality;

h. for loops;

i. parallel for loops;

j. parallel for synchronization statements;

k. deletion of variables;

l. display of variables (strings and scalars or matrices of either doubles or integers) ;

m. comments.

3.1.1.2. The SPINACH if, for and parallel-for statements **shall** support bodies with an arbitrary number of lines.

3.1.1.3. **Shall** check syntax validity of user inputs and report syntax errors.

3.1.1.4. **Shall** build up abstract syntax trees containing various elements after processing syntactically correct user inputs.

3.1.1.5. **Shall** support interpreting a set of plot commands for plotting team to plot data sets and configure plots.

3.1.1.6. **Shall** document SPINACH input language syntax and error messages on web pages. The documentation **shall** have a table of contents.

3.1.2. Interpreter Core Team

3.1.2.1. **Shall** support executing non-parallel SPINACH syntax commands stored in abstract syntax trees including:

a. creation of variables (strings and scalars or matrices of either doubles or integers);

b. creation of structures;

c. assignment;

d. support for non recursive functions that can accept multiple values and return single value;

e. if statements;

f. tests for equality and inequality;

g. for loops;

h. deletion of variables;

i. display of variables (strings and scalars or matrices of either doubles or integers) ;

j. comments.

3.1.2.2. **Shall** support executing a set of plot commands for plotting team to plot data sets and configure plots.

3.1.2.3. **Shall** break down abstract syntax trees of parallel SPINACH commands including

a. matrix operations (multiplication, addition, subtraction, transpose); scalar operations (multiplication, addition, subtraction, dot product),

b. parallel for loops,

c. parallel for synchronization statements

into multiple parallelized abstract syntax trees, then write each parallelized tree into some data stream (byte code?) and save it in the swarm memory.

3.1.2.4. **Shall** request swarm to execute parallel SPINACH commands when data stream is stored in the swarm memory. **Shall** assemble the results of the paralleled calculation one swarm computing is done.

3.1.2.5. **Shall** use hash map and/or other proper data structures to store variables.

3.1.2.6. **Shall** report semantic errors if happens.

3.1.2.7. **Shall** document SPINACH input language semantics and error messages on web pages. The documentation **shall** have a table of contents.

3.1.3. Swarm Computing Team

3.1.3.1. **Shall** be able to map each computer's username with its IP address and port. **Shall** allow user to join the swarm by supplying the IP address and port of any one of the computers in the swarm. Each computer in the swarm **shall** know the username, IP address, and port of all other computers in the swarm.

3.1.3.2. Each program (i.e. a set of source code) **shall** be owned by one user. The owner **shall** be able to grant read or read/write privileges to peers in the swarm. Ownership of a program **shall** be transferrable.

3.1.3.3. **Shall** be able to read data stream store by core module in the swarm memory and translate it into abstract syntax trees and other data.

3.1.3.4. **Shall** use the master-backup model demonstrated in the Architectural Concept Document to build up the collaborating environment.

3.1.3.5. **Shall** keep swarm memory for a particular program synchronized among all peers.

3.1.3.6. **Shall** notify the core module once swarm computing is completed and swarm memory is filled with partial results.

3.1.3.7. **Shall** interpret parallel-for synchronization statement as a barrier synchronization (i.e. forcing all threads in the parallel-for to arrive at the synchronization point before continuing).

3.1.3.8. **Shall** have the ability to continue computation of a program when an arbitrary computer in the swarm is disconnected without notice.

3.1.3.9. **Shall** allow multiple programs to be run at one time within the swarm. Each program **shall** have its own swarm memory.

3.1.3.10. **Shall** support chatting.

3.1.4. User Interface Team

3.1.4.1. **Shall** support connecting/disconnecting to/from a swarm.

3.1.4.2. **Shall** have a field in the main interface to view swarm member parameters (username, IP address, port, latency).

3.1.4.3. **Shall** have a field in the main interface for chatting with all users connected to the swarm.

3.1.4.4. **Shall** support creating a new program that has a separate window with text editor, control field, and result field. **Shall** supporting running multiple programs.

3.1.4.5. The source code text editor in each program window **shall** support:

a. optional line numbers
b. horizontal and vertical scroll bars
c. syntax highlighting

e. an optional line wrap

f. instant change reflected from all collaborators having write privilege if connected to a swarm (e.g. http://etherpad.com)

g. optional text highlighting with one distinct color per collaborator

3.1.4.6. The result field in each program window **shall** support viewing variables, results, and plots during and after execution. **Shall** decide how variables and results are displayed.

3.1.4.7. The control field in each program window **shall** support saving plots to a bitmap file, saving and loading program code text into text file.

3.1.4.8. The control field in each program window **shall** also support granting read or read/write privileges, and transferring ownership to a peer collaborator by entering username, or IP address and port. The user receiving privilege of a program **shall** see a new program window popped up containing same information of the program as all other peers'.

3.1.4.9. Any collaborator with write access permissions for a specific program **shall** be able to modify the program any time it is not running.

3.1.5. Plotting Team

3.1.5.1. **Shall** define a set of plotting commands that supports:

a. plotting single and multiple data sets on one program window

b. setting axis and plot titles

c. default axis scaling and user customizable axis scaling

d. using linear and logarithmic scaling

e. 3 dimensional data

3.1.5.2. **Shall** decide which data sets can be plotted (matrix, matrix multiplication, etc.). **Shall** notify Interpreter Core Team the data sets accepted by plotting commands so Interpreter Core Team can verify the semantics of plotting commands.

3.1.5.3. **Shall** decide how plotting command accepted data sets should be plotted so that user gets a visual representation of program data.

3.1.5.4. **Shall** support following modes for 3 dimensional data plotting:

a. "terrain" mode

b. vector field mode

c. mode where there is a 2D image but the color of a pixel shows the height of the Z axis

3.1.5.5. **Shall** support saving plots to a bitmap file

3.1.5.6. **Shall** use OpenGL and/or other technologies for plotting.

3.1.6. Test Team

3.1.6.1. **Shall** design the logo for SPINACH

3.1.6.2. **Shall** document Qualification Test procedure and results of all SPINACH features. The documentation **shall** have a table of contents

3.1.6.3. **Shall** document a getting started tutorial that walks the user through an example application that tests every part of the core SPINACH syntax

3.2 Process Requirements

These requirements specify the physical structure of delivered code and the environment where it must operate.

3.2.1 Physical Structures

3.2.1.1 The SPINA source code **shall** be composed of modules.
3.2.1.2 SPINA **shall** be implemented using Visual C# .Net.
3.2.1.3 The user interfaces **shall** delegate all operation, not directly associated with providing the user interfaces, to server modules.
3.2.1.4 All modules **shall** be provided with manual pages and correct maintenance pages
3.2.1.5 Each server module **shall** be provided with a test configuration

3.2.2 Development Environment

3.2.2.1 The SPINA System **shall** build and operate in the ECS clusters, e.g., Link 010, 202, 201, 274 or CST 2-112