Concrete Architecture of LEAP Database Management System

Analyzed by

BASEM SHIHADA

University of Waterloo
Dept. of Computer Science
200 University Avenue West
Waterloo, Ontario, Canada
(519) 8851211 ext. 5387

CS798 Assignment #2

19th February 2002

Table of Contents

1 Abstract

Developers of Open source software requires high-level component description, inter-component relation description and a detailed component description. Most of the current open source implementations suffer form the availability of a comprehensive description of the software component. This results a miss-concept in the architecture of the software or a future mistakes in determining how the software components communicate between each other. Furthermore, by the growth of the software, the developer becomes unable to identify or to locate the software components relations, which will make the future modification sometimes very expensive. This results that some open source software goes to end.

This assignment presents a concrete architecture for the open source LEAP Database Management System. "LEAP is a relational database management system (RDBMS). It is used as an educational tool around the world to help students, and assist both researchers and teachers as they study or teach databases" originally written as an undergraduate project by Richard Leyton. LEAP is written in C with an approximate 30,100 line of code. LEAP is a full open source project: It's free to download; free to use; and anyone has full access to the source code; and can participate with whatever found its approbriate to enhance its features..

The goal of this assignment is to go through the low-level functionality of LEAP and build a concrete architecture to this DBMS. This assignment discusses the software major components "sub-systems", the component relations and the whole picture of the software architecture. In this assignment Rigi has been used as a reverse engineering tool.

2 Introduction

LEAP is a relational database management system, designed as an interpreter for the relational algebra. It represents data as being stored in tables also called *relations*. A raw in the relation is called *tuple*. Columns in a relation are called *attributes*. As a certain tuple is need from the relation, a tuple is selected depends on an identifier value, a new operation is created and a tuple is extracted from the relation. This operation is called the *select* operation, which based on a Structured Query Language (SQL). The general conceptual components of the relational database are, the user application software, the database management system and the actual database (file system stored in the physical device).

Within LEAP, the various relations are divided into separate databases. A database is a collection of logically related relations. The relations within a database are usually related to one another. LEAP's databases are relatively simple, and there are two special databases: master and user. Master contains the data dictionary. User is the default database the user connects to. Also the standard LEAP distribution software contains three additional databases named as follows: date, stanczyk and korth.

This assignment is outlined as follows; section 3 starts with a discussion of the used reverse engineering toolkit (Rigi). Moving directly to cover the sub-system components and how they related to each other. Then focuses on LEAP’s modules, with a description of each module and draw a concrete architecture of each module, graphs will be available to illustrate the active functions. Within section 3 specifically an overall system graph is given. Section 4 illustrates two different scenarios to show how LEAP’s components interact with each other. Section 5 lists some word abbreviations and important definitions. Section 6 mainly shows what LEAP’s developers are planning to do to enhance LEAP for the future use and shows the level of flexibility for future updates.

3 Concrete Architecture

3.1 Reverse Engineering Toolkit Rigi

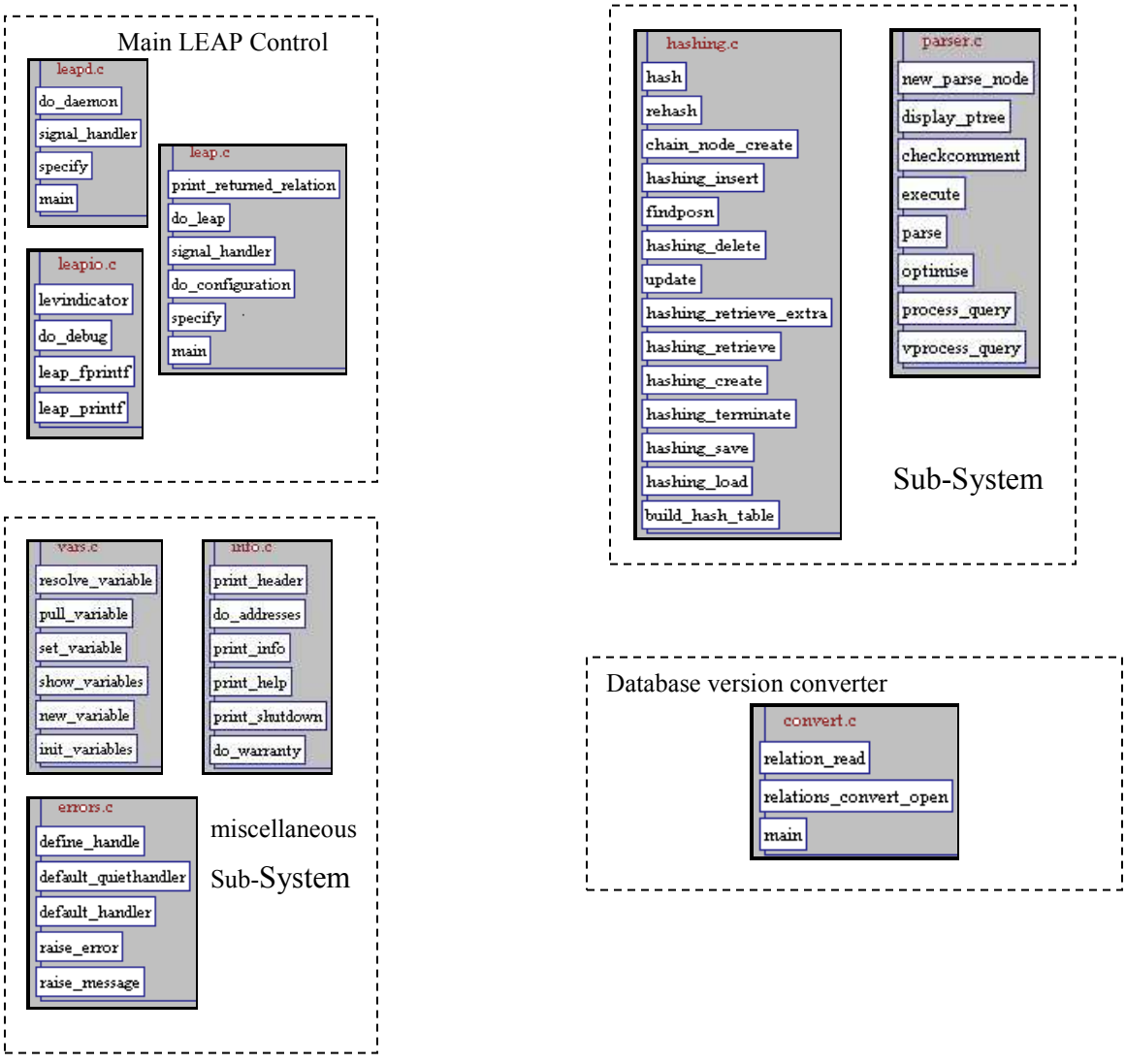
The Rigi reverse engineering tool was developed as team led by Prof. H.A. Müller of University of Victoria. It's intended to help developers analyze and summarize the structures of legacy software systems. Therefore, this toolkit is considered one suitable reverse engineering toolkit for attempting to understand large, complex systems when there may not be adequate documentation or knowledge available. It's also intended to be used to understand legacy systems.

The Rigi tool consists of several sub-tools – RigiEdit and RigiParse. Cparse, Rigiparse, and sortrsf are all utilities provided with Rigi to automate the parsing tasks. The following is a brief description of the two main components of Rigi:

- RigiEdit: This is the graph editor, which can be used to browse, analyze, and modify the graph that models the software system. The graph hierarchically clusters related entities into subsystems, which are then clustered into larger subsystems [6].
- RigiParse: This program is independent from RigiEdit. It's Rigi's parser which extracts entities and the relationships between the entities from the given source code. For example, the C parser extracts C functions and C structure definitions as the entities. The relationships that are documented by the parser are the function calls between function entities and data references between function and data structure entities [6].

3.2 LEAP's Sub-systems Graph

This section graphs each independent sub-system. Figure 1 graphs every leap file and its contents such as, functions; these files are grouped together depending on their functionality into a bigger sub-system. Depending on the common functionality studded from leapd.c, leap.c, and leapio.c they are categorized into one sub-system called main leap control sub-system. rtional.c, dbase.c, database.c cond.c and attribs.c are grouped together under database sub-system. Miscellaneous sub-system is mainly used by every sub-system; this sub-system consists of vars.c, info.c, and errors.c. Leap parser sub-system, which consist of the parser, the stack and the hashing function. These are implemented in parser.c and hashing.c. Database version converter is an independent sub-system, which used to convert the database system from verstion 1.0 to verstion 1.1. A leap main functionality is implemented in the utility sub-system; this sub-system is accessed and called by all the fuctions, which need any access to the database through the operating system. Every function that deals with the operating system interface is implemented in the utili.c sub-system.



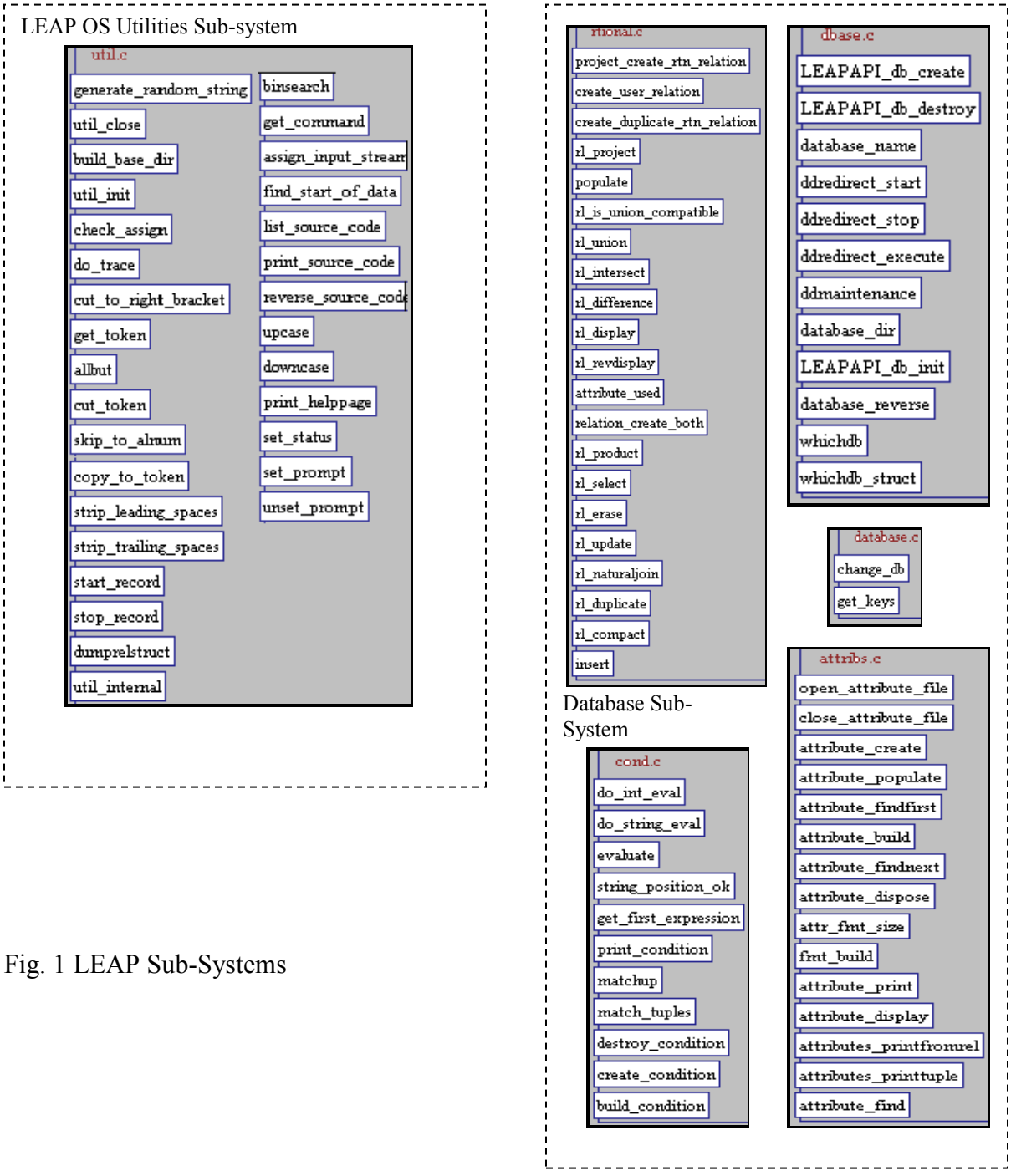


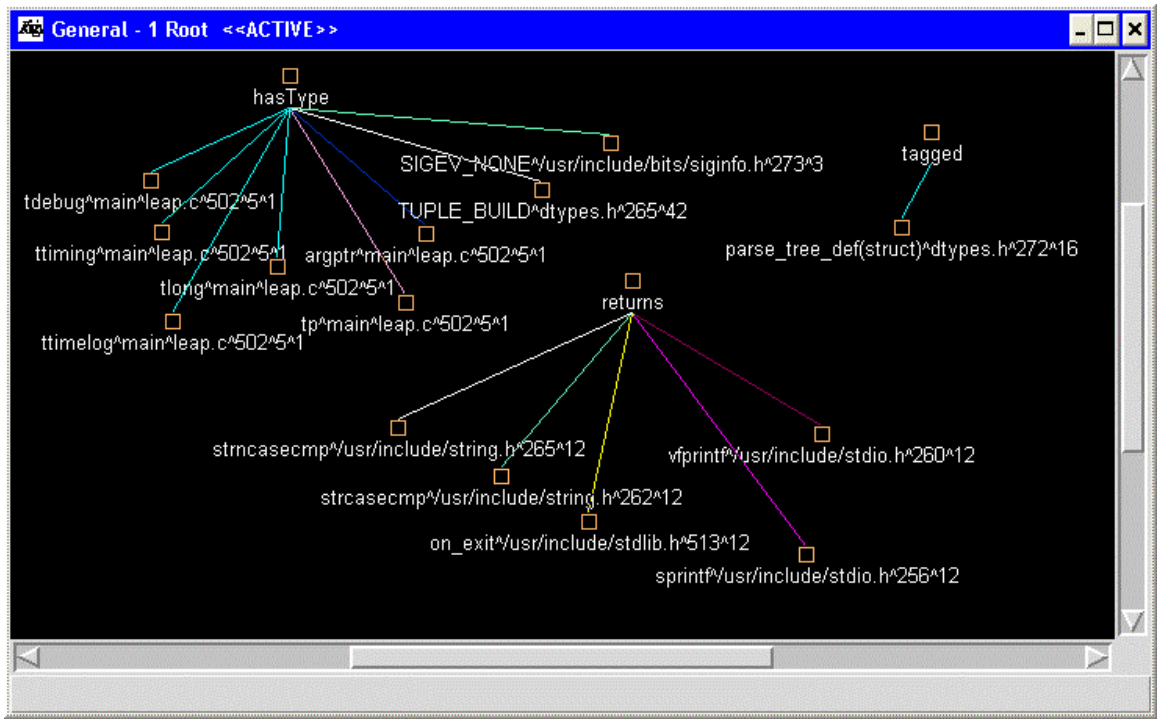
Fig. 1 LEAP Sub-Systems

3.3 LEAP's Components Illustration

3.3.1 Leap main file

Main LEAP loop and functions. This component is responsible of performing the display of LEAP title, and processes the command line. Clean LEAP termination also should occur here. Any other functions terminations are done everywhere else is "unclean", and should return an error status. Define error status in the dtypes.h file.
The following is a list of the declared routines and their functionality.

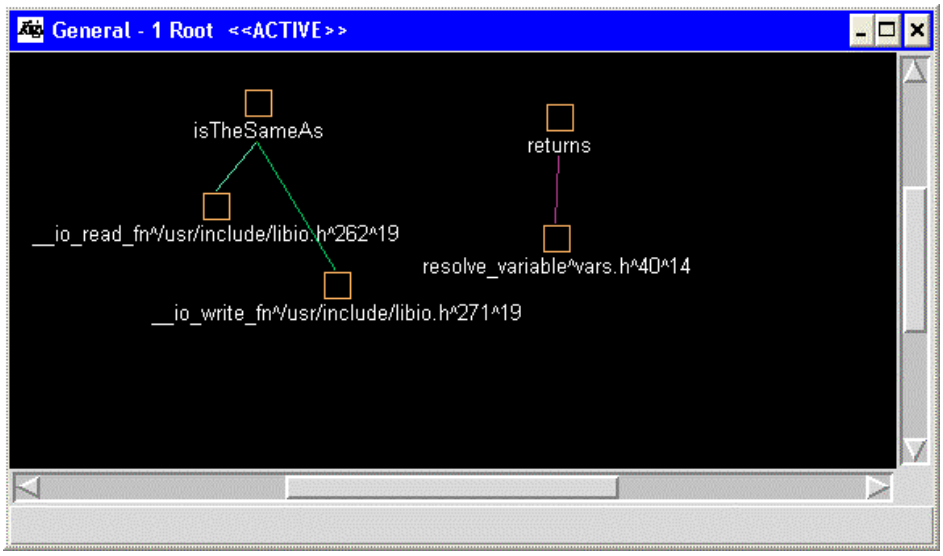
- print_returned_relation: Displays the relation name. If a relation has been returned, it will be buffered as a string and has been returned.
- do_leap: Main LEAP routine which contains the user interface , and calls other routines.
- signal_handler: Signal handler for signals requiring shutdown, hang-up, and termination.
- do_configuration: Configure the LEAP setup.



3.3.2 Leap I/O

Main Leap Input and Output operations.

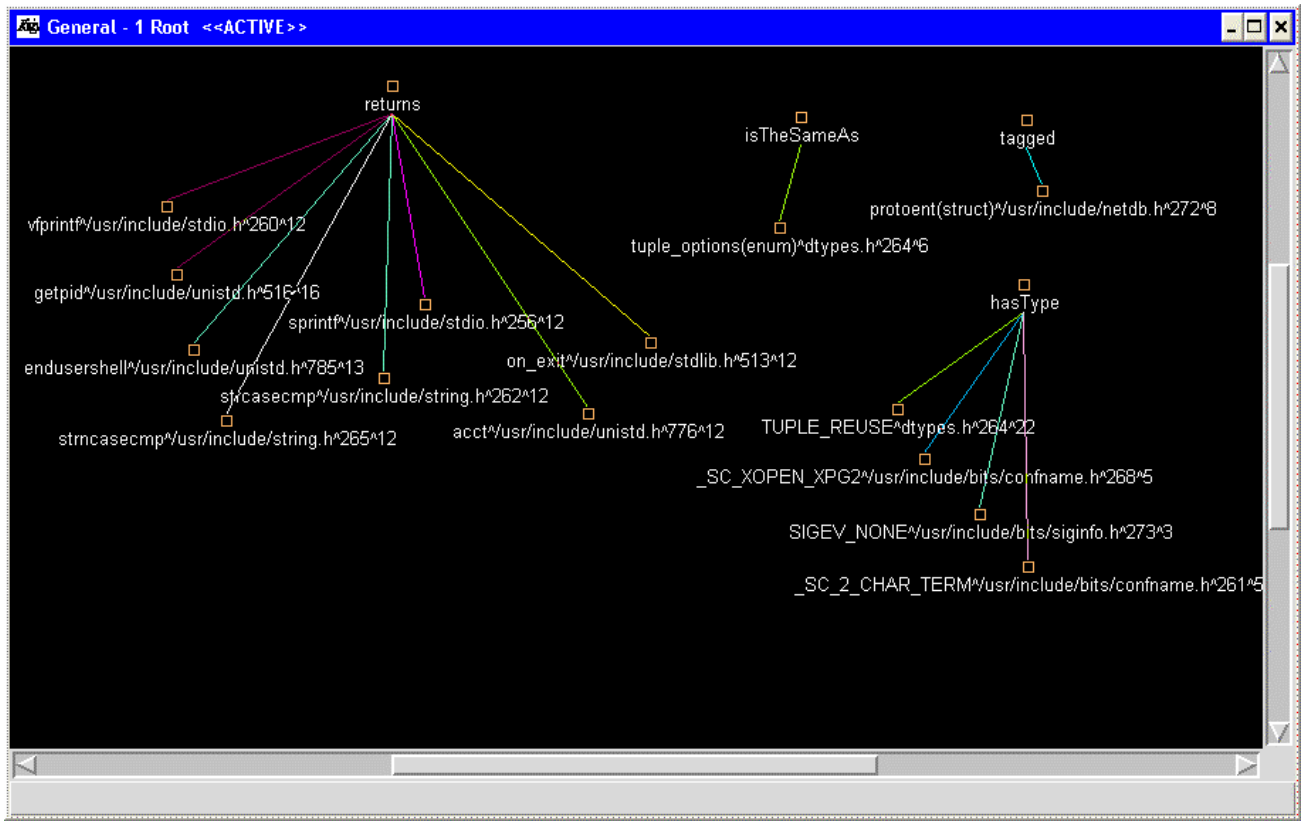
- levindicator: Toggle the debug level indicator used in high level debugging.
- leap_debug: Debug message, printed only if debug level is at or above parameter.



3.2.3 leapd

Main LEAP loop which has all functions for socket handling. In this file the following operations are done, open the master database, open up the default user database, check to see if the logins relation exists, and build the LEAP logins relation.

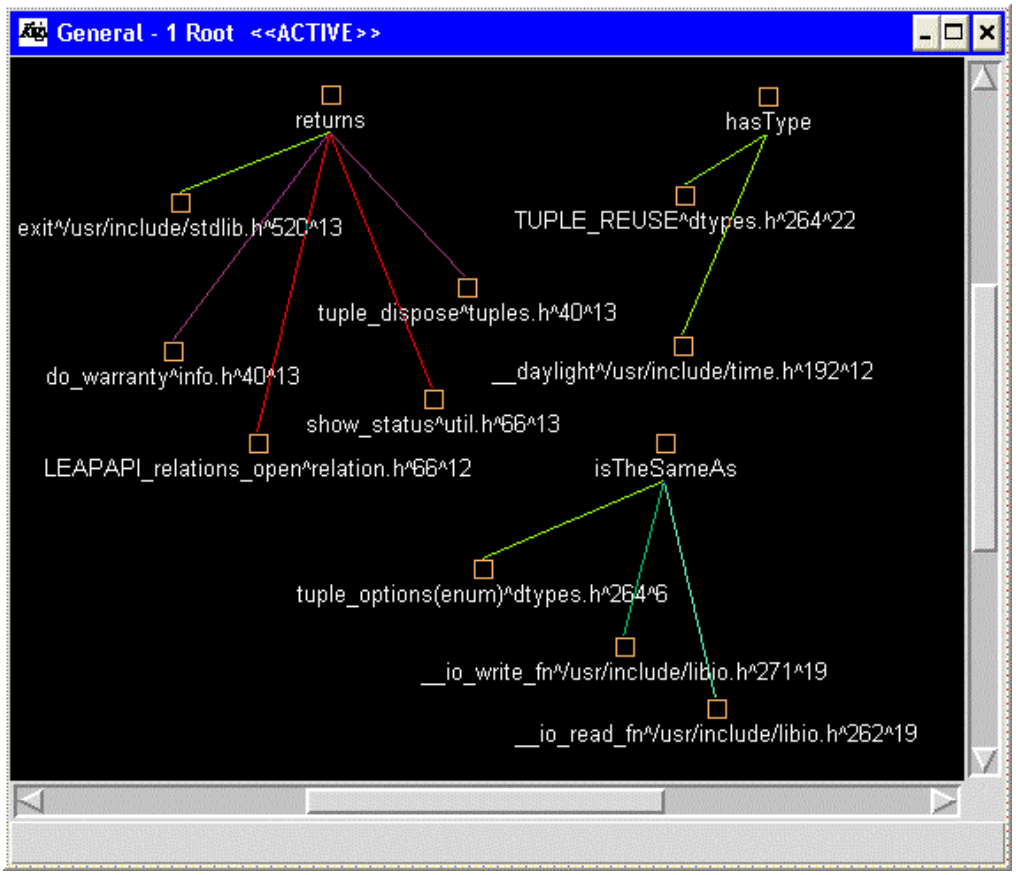
- do_daemon: socket handling.
- signal_handler: signal handler for signals requiring shutdown - Hang-up, termination.



3.3.3 Parser

Parser module functions take a user command string, and build a parse tree, then execute it.

- new_parse_node: Create a new parse node.
- display_ptree: if a debug level is set, a parse tree is displayed.
- execute: This function takes a parse tree, and applies the operations as appropriate, in the correct order, iteratively.
- parse: Takes user command string, and processes it as an expression, returning a relation, or null if no results occurred. If an error occurs, then check error codes in global variables for this error.
- optimize: Performs an optimization sweep of the parse tree.
- process_query: High-level control routine for processing a query.
- vprocess_query: Allows variable length query processing.

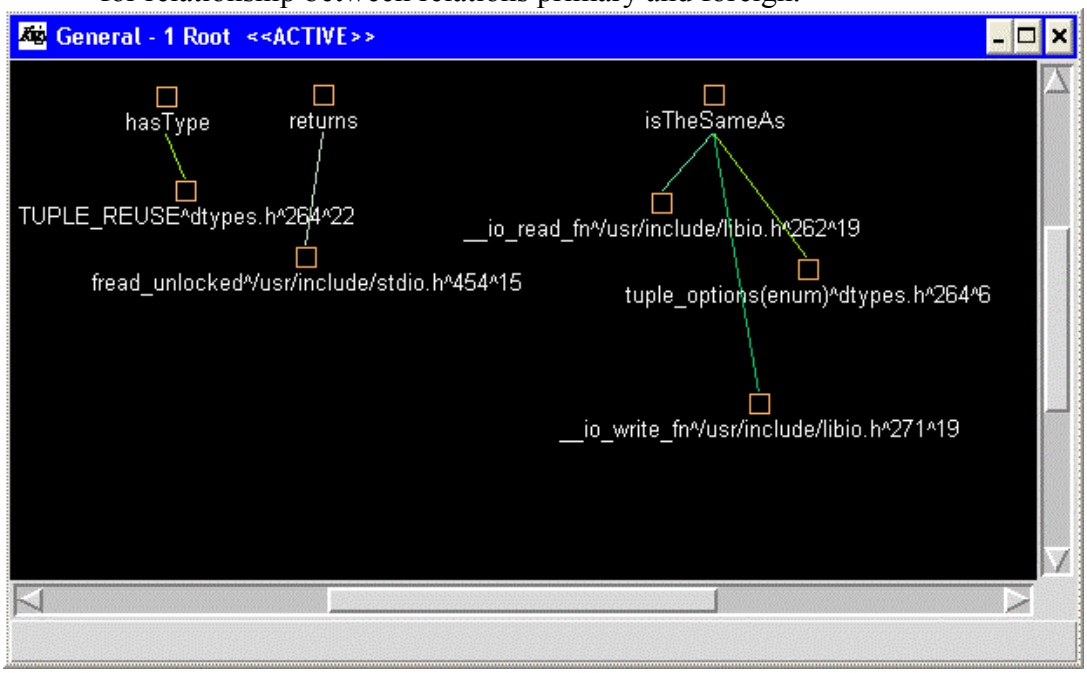


3.3.4 Database

Database management functions.

- change_db: Changes the current database to the one specified by the user.

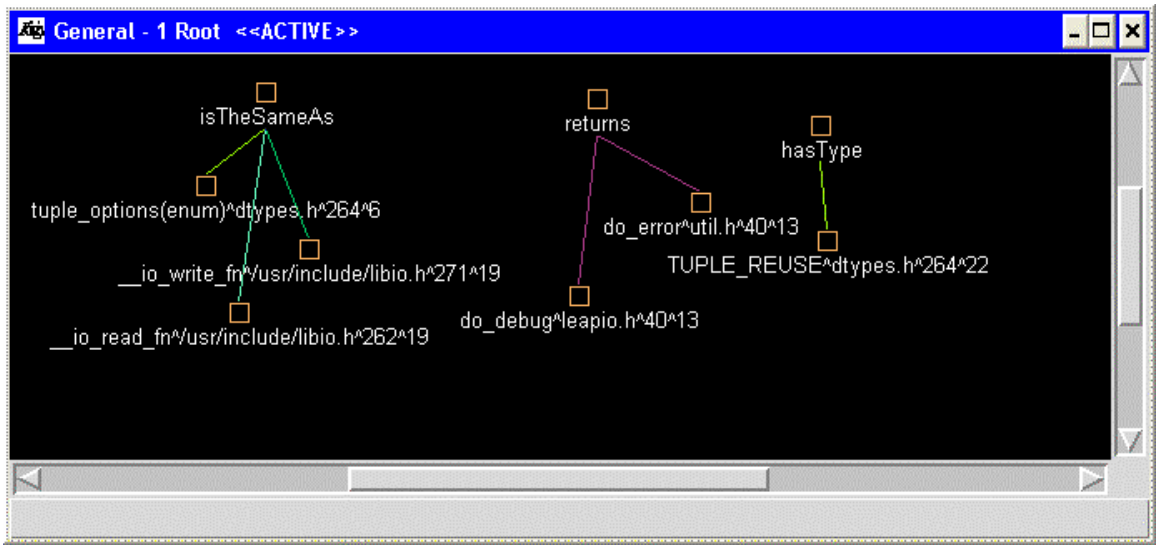
- get_keys: Fetches list of keys into strings pointed to by foreign/primary names, for relationship between relations primary and foreign.



3.3.5 Dbase

Main database structure access/modification functions.

- LEAPAPI_db_create: API routine to create a database structure on a specific database. Make sure the database name does not exceed the size of a directory name.
- LEAPAPI_db_destroy: This releases the memory of the database. Used by the end of the database life.
- database_name: Returns the pointer to the database name string.
- ddredirect_start: start the redirection for data dictionary maintenance.
- ddredirect_stop: Stop the redirection for data dictionary maintenance.
- ddredirect_execute: Execute the data dictionary maintenance redirected.
- Ddmaintenance: Depending on whether data dictionary maintenance is occurring, execute a query, or store commands in a file.
- database_dir: Returns pointer to string containing the database directory.
- LEAPAPI_db_init: create an external and internal database structure.
- ddredirect_execute: Maintain static relations and update the leap types relation.
- database_reverse: Reverse engineer a database.

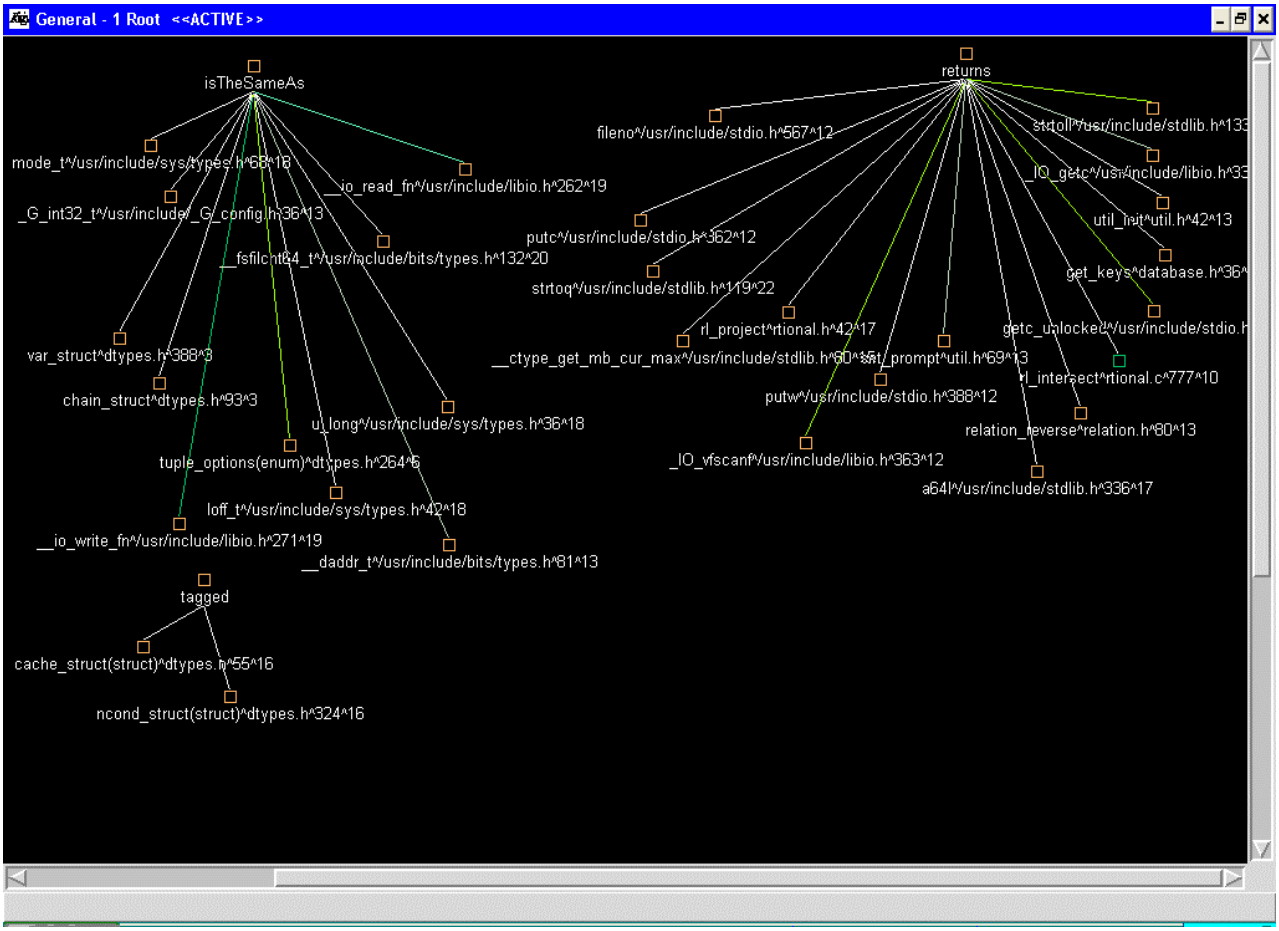


3.3.6 Relation

Controls the basic relation structure.

- delete_relation: Delete the files associated with a given relation. It first gets the base directory of a relation, then determine which file should be deleted.
- relation_insert: Insert the relation into the database relation structure. It first checks the insert is going into the right db, then search the structure for the position at which the insertion should take place.
- create_tempfile: Create a temporary "tag" file
- remove_tempfile: remove temporary file.
- relation_change: Change status of a relation from relation name to temp.
- relation_create: Create the specified relation, and insert it into the database structure.
- relation_print: Print out basic information about the relation.
- relation_find: Locates the specified relation in the relation structure.
- relation_display: Displays all of the relations in the database structure.
- relation_full_path: Gets the relations full path (including the name, but not the extension.
- relation_new_read: Read a specified relation into memory, and return a pointer to it. Only works with new format relations.
- relations_open:
- relations_ddopen: Load all of the relations in the specified database according to data dictionary definitions.
- LEAPAPI_relations_open: check whether the relation is open or no.
- relation_dispose: Dispose of a relation's memory.
- relation_dispose: Dispose of a relation.
- relations_dispose_all: Dispose of all relations in a relation structure.
- relation_remove: Remove a relation from the relation structure.
- relation_rename: Rename specified relation, or specified attribute.
- relation_revattrib: Print out the information contained in the specified attribute in a reverse form that can be embedded in create database command.

- relation_reverse: Print out all of the attributes associated with a relation in a reversible format.

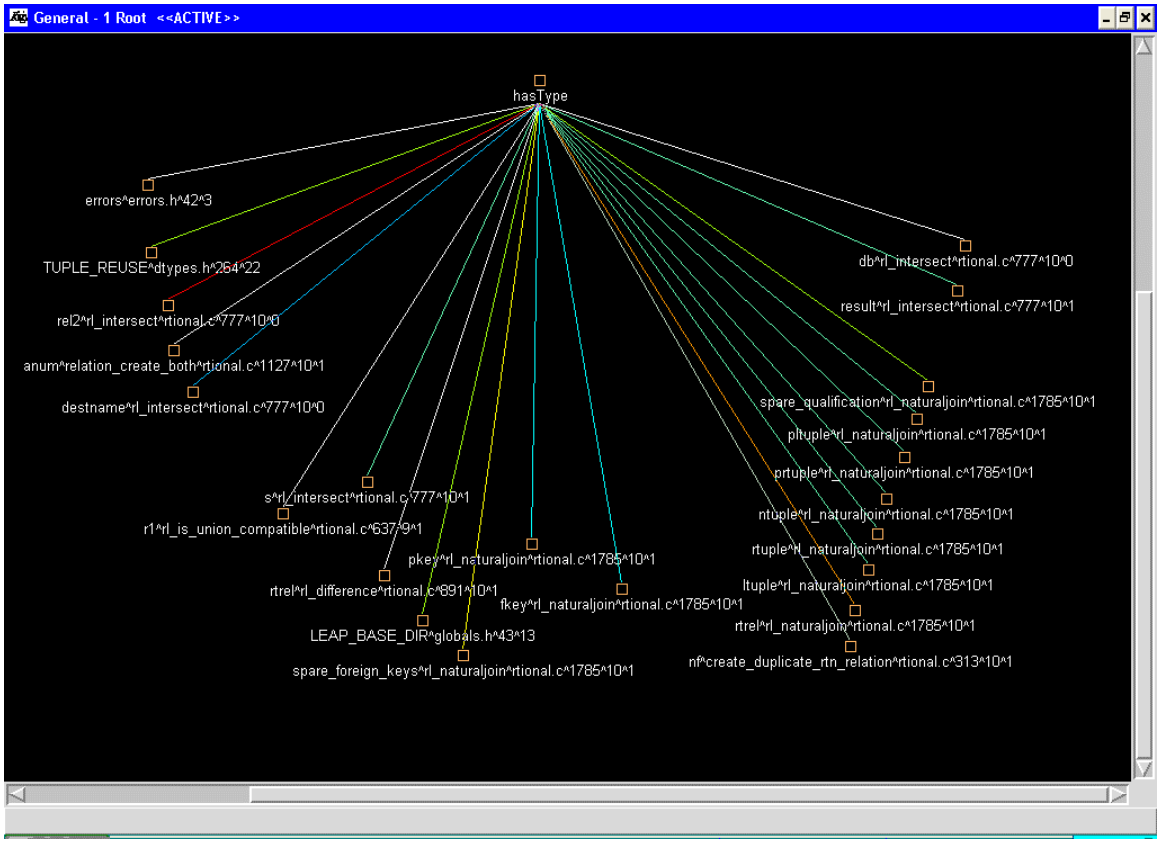


3.3.7 Rtional

Relational Operators.

- project_create_rtn_relation: Creates an (empty) relation with the specified attributes only.
- create_user_relation: Creates a relation for the user according to schema in attrib_list of format ((name1,type1,length1) ...).
- create_duplicate_rtn_relation: Create a relation with the same attribute names as specified relation.
- rl_project: Returns the result of performing a relation project operation on the specified relation, taking the attributes specified in attribs.
- rl_is_union_compatible: Checks if the specified relations are union compatible.
- rl_union: Performs the algebraic operation union with the specified relations
- rl_intersect: Intersection operator.
- rl_difference: Difference operator - based extensively on intersect.

- rl_display: Display the tuples in a relation.
- rl_display: Display the tuples in a relation for reverse engineering.
- attribute_used: Returns ture If the attribute specified in attrib is located in the "string" array.
- relation_create_both: Creates a new relation that contains the attributes from two source relations.
- rl_product: Produces the cartesian product of the two specified relations
- rl_select: The relational select/restrict operator
- rl_erase: The delete tuple operator (rl_delete is used in the read line library.
- rl_update: essentially the select operator to identify the tuple, but then some balls.
- rl_naturaljoin: The new join operator. Returns null if fails, or pointer to a new relation
- rl_duplicate: Creates a duplicate relation.
- Insert: Performs an insert of data in the data base field.

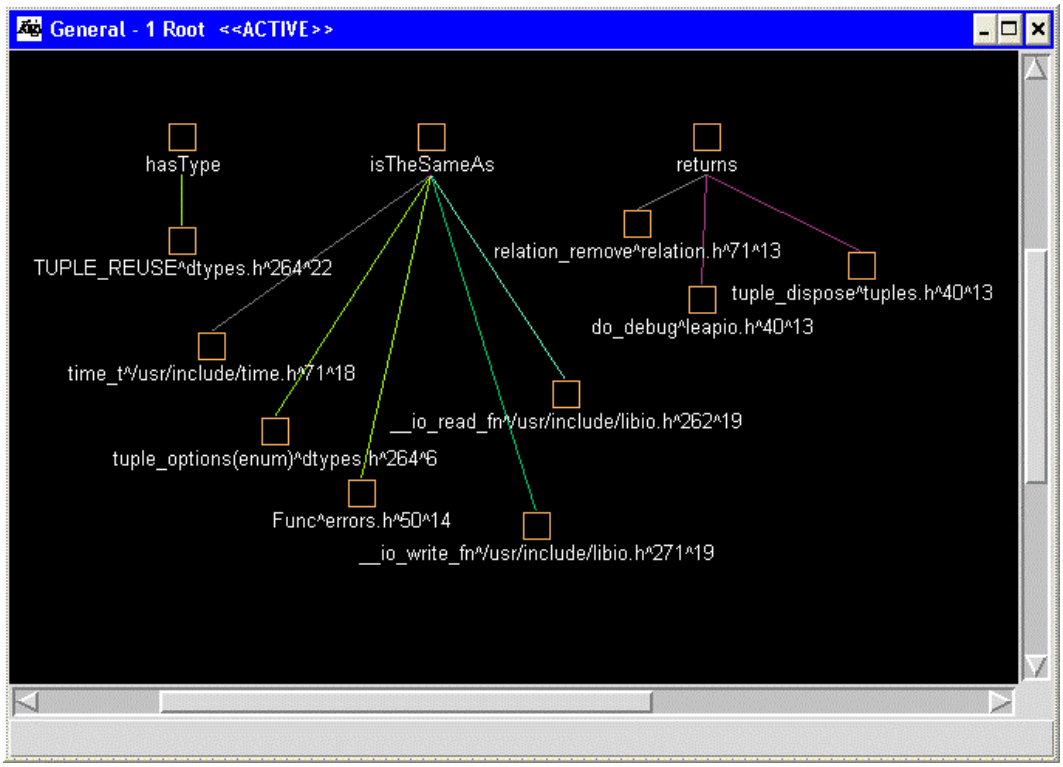


3.3.8 Attribs

Attribute Functions - Access/Modification of attributes in a relation.

- open_attribute_file: Opens the attribute file, and returns a database file pointer.
- close_attribute_file: Close the specified file.
- attribute_create (Same as field_create): Create an attribute for a relation.
- attribute_populate: Reads from the specified file an attribute, and populates the specified attribute with the data read. Returns the result of the last read.

- attribute_findfirst: Return the first attribute in a relation
- attribute_build: Builds an attribute from parameters
- attribute_findnext: Locates the next attribute, and optionally creates a new node for this. In addition, if the last node is located, it is optional whether the node is disposed. (This is all in order to populate a tuple structure).
- attribute_dispose: Dispose the memory location used by the specified attribute, and close files accordingly.
- fmt_build: Builds a format string for printing out an attribute or value of specified size.
- attribute_print: Print out the information contained in the specified attribute.
- attribute_display: Print out all of the attributes associated with a relation.
- attribute_print: Print out all of the attributes associated with a relation - For rl_display mainly.
- attribute_print: Print out all of the attributes associated with a relation - For rl_display mainly.
- attribute_find: Locates the given attribute, and returns an attribute structure



3.3.9 Convert

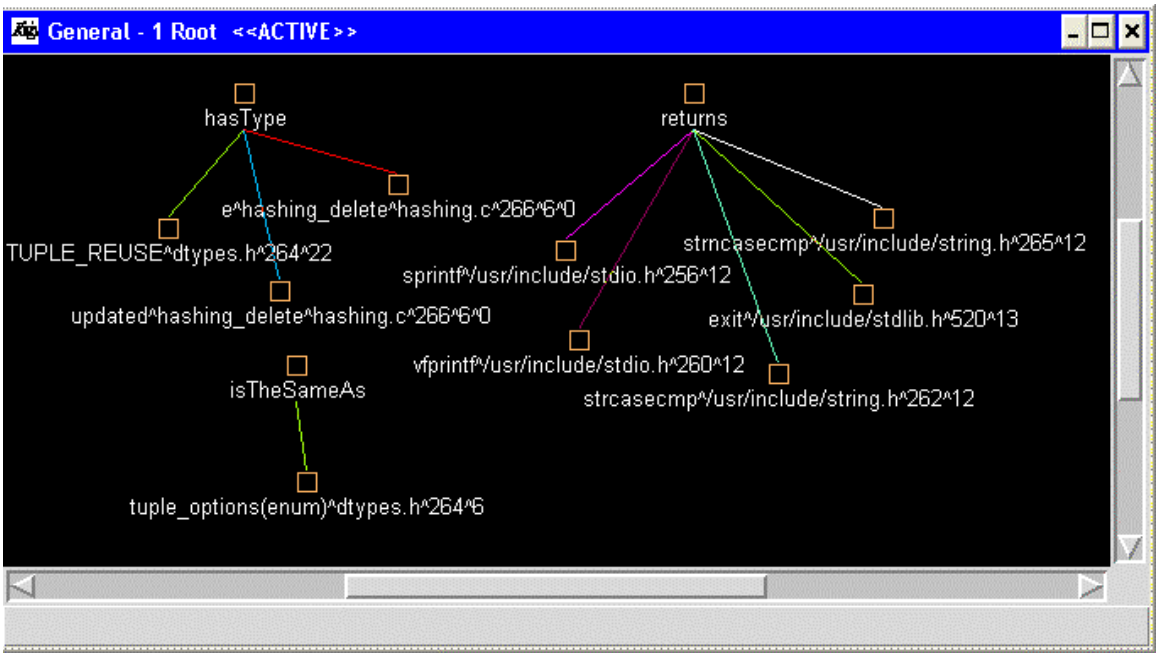
Convert database format from 1.0 to 1.1. This sub-system is independent from the DBMS

- relation_read: Read a specified relation into memory, and return a pointer to it.
- relations_open: Load all of the relations in the specified database.
- Main: main file, calls other functions

3.3.10 Hashing

Hashing functions.

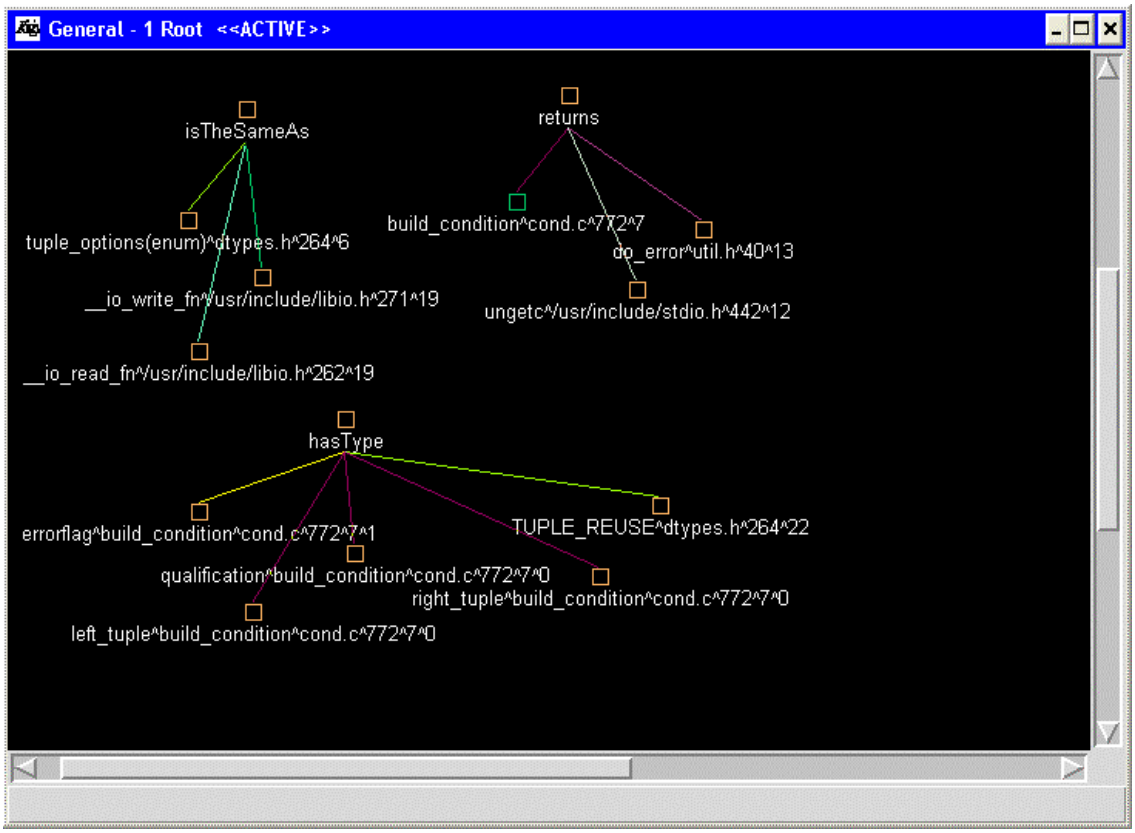
- hash: Returns a hash key, using folding.
- Rehash: rehash and return the result value.
- chain_node_create: Create a chain node.
- hashing_insert: Insert a key into the specified hashing table.
- Indposn: If HT contains an element with key value tkey, then found is true and posn is that element's position. Otherwise found is false. Locates a node in the specified hash table, and returns the position and boolean value for the search result.
- hashing_delete: Locate a given node in the hash table, and delete it.
- Update: Locate a given node in the hash table, and update its contents to those specified.
- hashing_retrieve_extra: Retrieve a given node (and extra values) from the hash table.
- hashing_retrieve: Retrieve a given node from the hash table.
- hashing_create: Create a new hashing table, and return a pointer to it.
- hashing_terminate: Dispose of the chains associated with a hashing table, and the hashing table itself.
- hashing_save: Saves the hashing table to the filename specified.
- hashing_load: Create a table, and load the hash file specified.
- build_hash_table: Build a hash table from a relation, and return a pointer.



3.3.11 Cond

Routines for evaluating conditions against tuples.

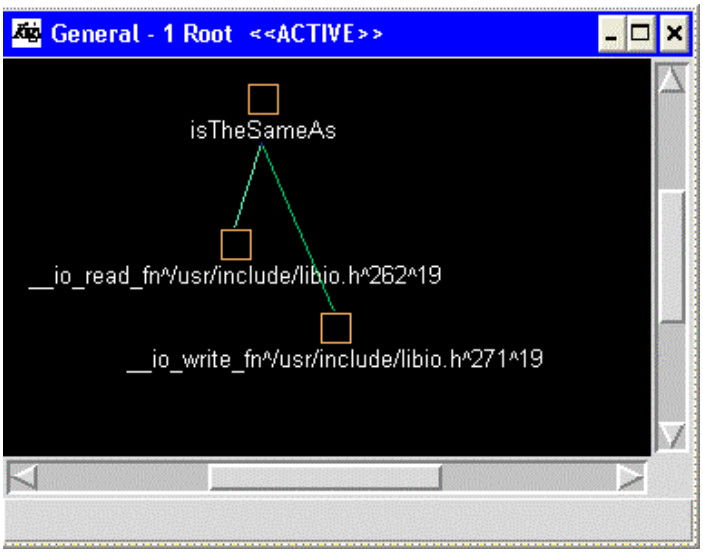
- do_int_eval: Performs an integer based evaluation of the data.
- do_string_eval: evaluate a string and return true or false.
- evaluate: Evaluates a condition based on condition structure, and specified tuple, and returns true or false.
- check_string_position: Check the current position (cpos) in the string (string_length), whilst evaluating text.
- get_first_expression: Takes an expression from the string specified (text), where a string is separated by boolean expressions in the boolean array.
- print_condition: Displays the condition structure.
- Matchup: Performs the work of matching the condition to the appropriate tuple, fetching pointers to data.
- match_tuples: Match up condition fields to tuples, so references can quickly be made when evaluating if a condition is true or false.
- destroy_condition: Dispose of a condition structure. Returns success.
- create_condition: prepare to create a new relation.
- build_condition: Takes a qualification, and builds a linked list structure with the qualification built. Return null If fails, pointer to first condition if success.



3.3.12 Globals

Global variables

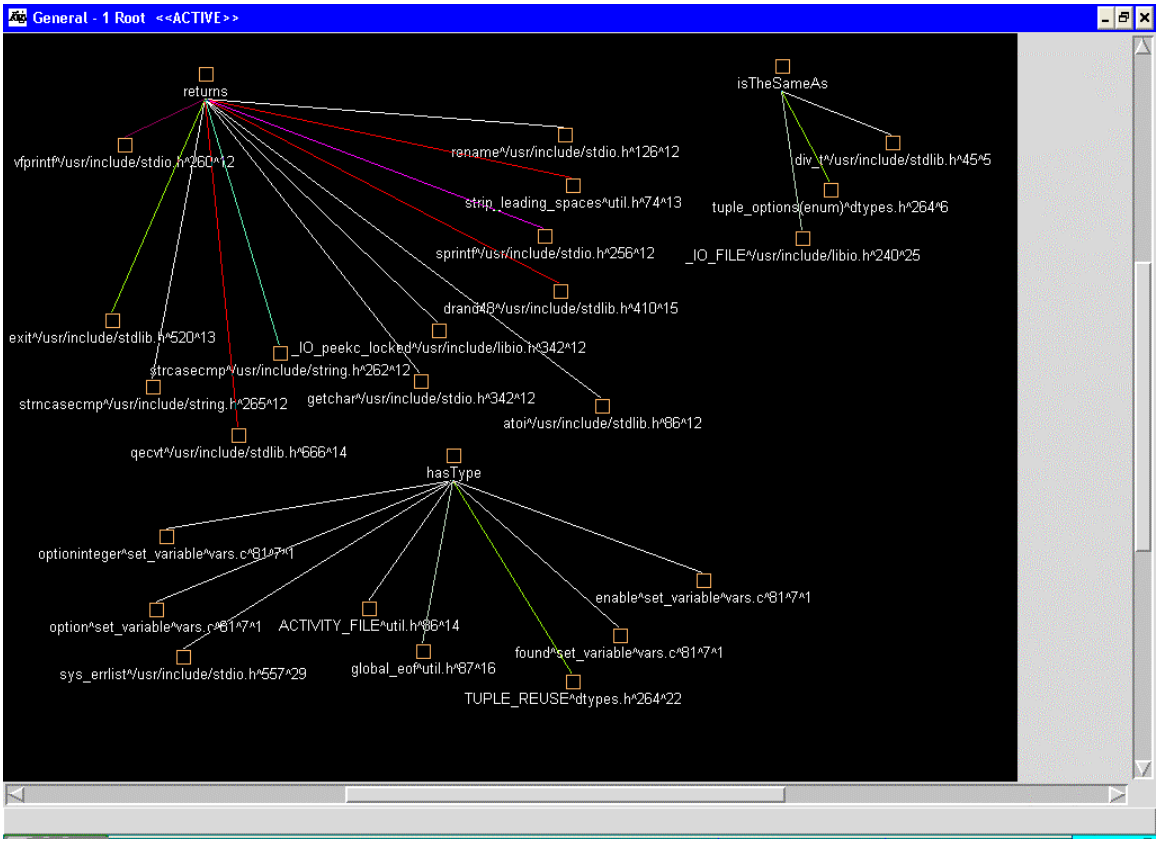
- Actual database's
- Boolean, when LEAP is on the way down.
- File, this is normally standard input, but the user might want to source a file
- LEAP internal version number.
- Boolean to determine updates to data dictionary.
- Configuration mode.
- Recording mode.
- Recording file pointer.
- Client socket.



3.3.13 Vars

Variable resolution/maintenance

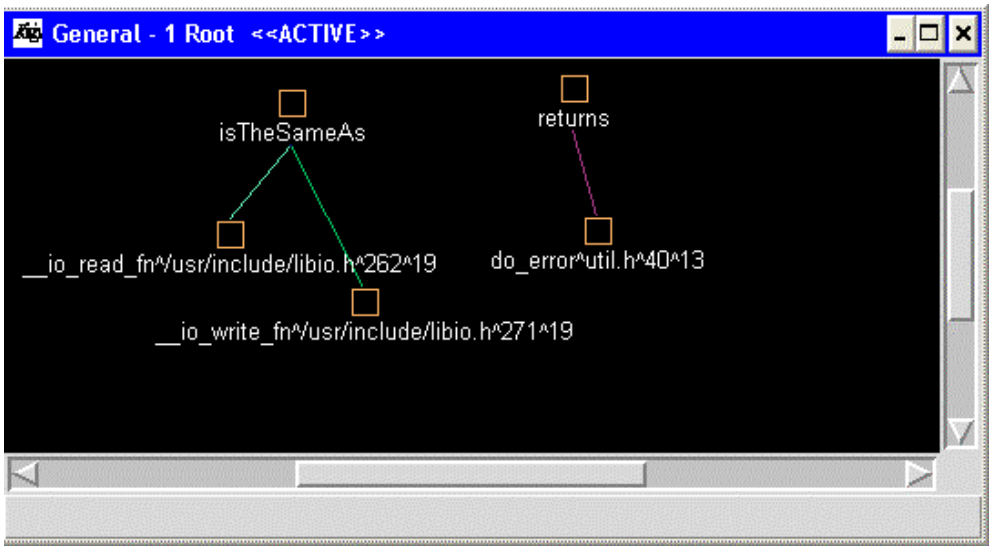
- resolve_variable: Resolves specified variable.
- pull_variable: Pulls out a specific variable number.
- set_variable: Specifies the given variable.
- show_variables: Shows settings of all variables.
- new_variable: Creates a new variable.
- init_variables: Initialize internal structures.



3.3.14 Errors

Handle errors. Define the error structure.

- define_handle: Definition of error handler.
- default_quiethandler: Switch on error # severity.
- default_handler: Switch on error # severity.
- raise_error: Raise an error message.
- raise_error: Raise an informational message.

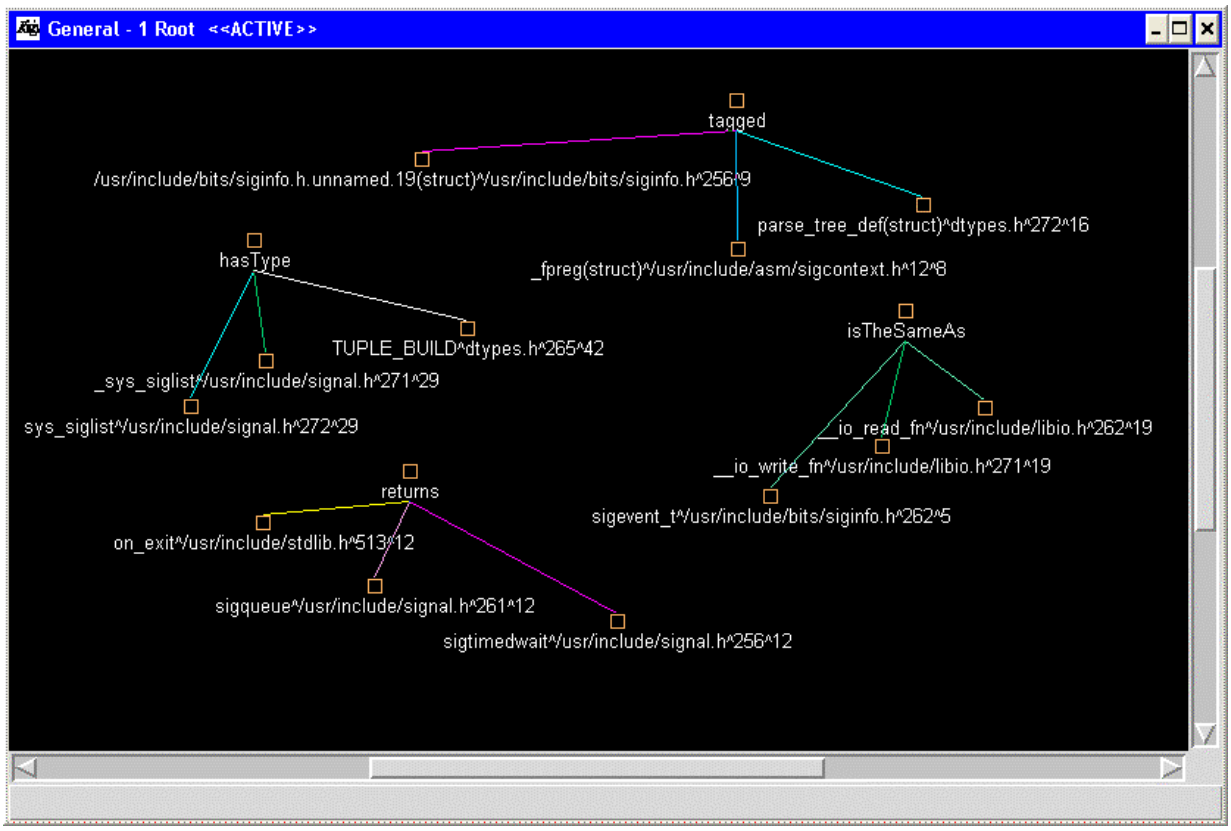


3.3.15 Utili

Utility Module - Contains all of the useful routines, and OS dependent functionalities.

- generate_random_string: Generates a string of 'size' random(ish) characters.
- util_close: Close files and other routines.
- build_base_dir: Build the global variables for the base directory. Putting this in a routine makes sense, so we can try various options if a file is not found.
- util_init: This routine sets up any global variables for the utilities unit, for example, the full path to the errors file, to save work later.
- check_assign: Checks whether a pointer is assigned or not. If not, then memory has probably run out.
- do_trace: Reports something to the screen, and if set, to the report file as well.
- cut_to_right_bracket: This should return a ptr to an expression contained within the brackets of depth in result and as result new for new parser. if force is true then force brackets onto the string, else don't - and return nothing if no brackets exist.
- get_token: Gets the first token from the string, and returns it (in result, and as a result).
- Allbut: return string with contents of string without any chars specified in chars
- cut_token: Cuts the first token from the string, and returns it (in result, and as a result).
- binsearch: apply binary search algorithm.
- get_command: whenever a command is needed its called.
- ssign_input_stream: This is used to assign the input stream (input_stream) declared as extern in dtypes, and in the main module, to the specified file. If empty, then the file is reset to stdin.
- find_start_of_data: Scans over a string, and locates the first text.
- list_source_code: Prints out the source files in a database.
- print_source_code: Prints the specified source file name. Assumes the .src extension is not specified.
- reverse_source_code: Reverse out the source code in a directory.

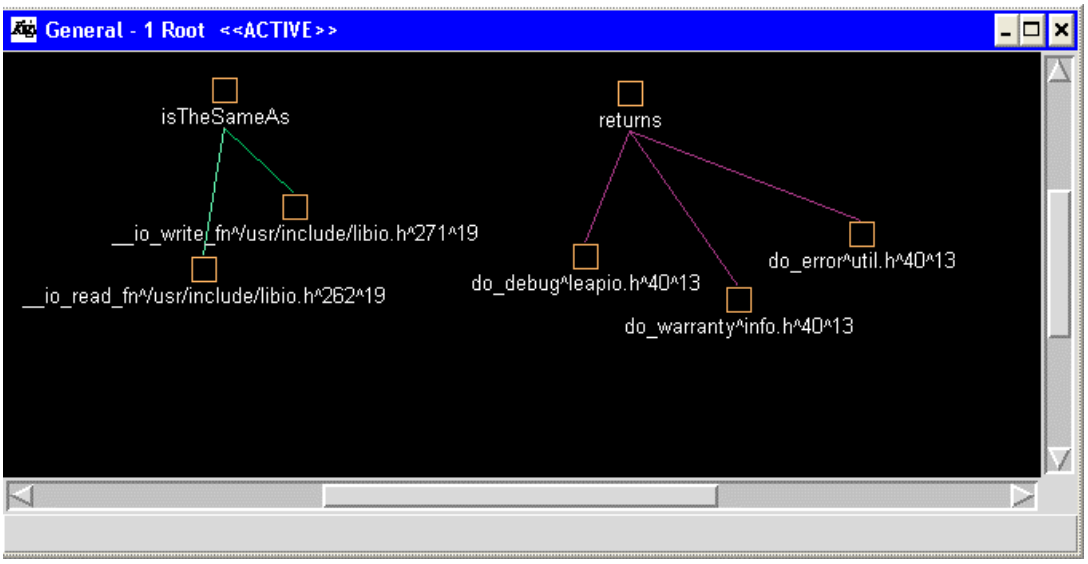
- upcase: Converts the specified string to upper case.
- down: Converts the specified string to lower case.
- print_helppage: Prints the help page specified, or the introduction/summary page in the absence of a value.
- set_status: Sets status flags on or off.
- set_prompt: Sets the prompt to that specified.
- unset_prompt: Sets the prompt back to the default.
- copy_to_token: Tokenises source using strtok, and copies to destination.
- strip_leading_spaces: Strips the leading spaces from source.
- strip_trailing_spaces: strips out trailing spaces in string.
- start_record: Build the directory path, and open the file.
- util_internal: Display internal LEAP structures - this will be expanded to provide a mechanism to probe internal structures in due course.



3.3.16 Info

Information routines.

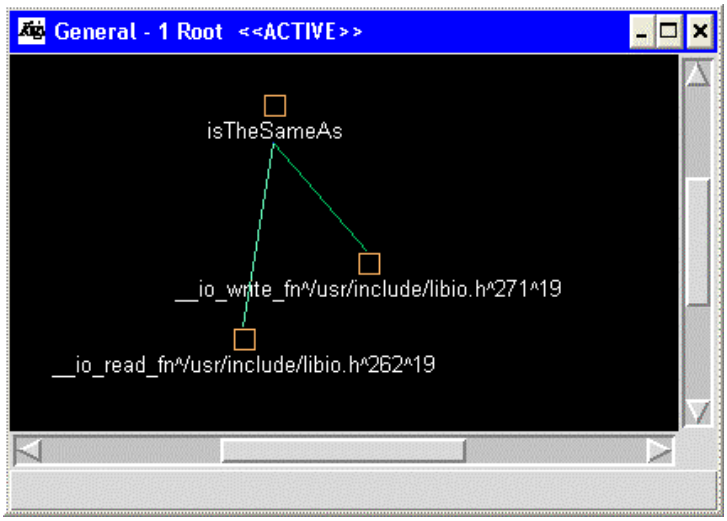
- print_version: Prints the LEAP version header.
- do_addresses: displays contact information.
- print_info: displays the warranty and conditions blurb.
- print_help: Print command line help and so on.
- print_shutdown: Print the shutdown message.
- do_warranty: Print the warrant information of LEAP.



3.3.17 P_Stake

Parser stack functions.

- ps_create_stack: Creates a stack structure.
- pt_push_stack: Push an item onto the stack.
- pt_pop_stack: Pop an item off of the stack.
- pt_stack_dispose: Dispose of a stack structure.
- pt_flush_stack: Dispose of ALL Items in a stack structure.



3.3 LEAP's Sub-Systems Relations

From the low level functionality of LEAP, there are many common scenarios for sub-system to sub-system interactions. This section will illustrate the low-level functionalities of each sub-system and the overall interactions between the sub-systems. Figure 2 illustrate LEAP's sub-system modules.

Main Leap sub-system consists of leap.c, leapio.c, and leapd.c. This is the main LEAP program interface, which handles the user I/O commands, in other words it's the LEAP's command line processing unit. Other main important functionality is to build-up and start the connections between the modules. This sub-system calls every other sub-system and received a call from each.

Errors Handler "errors.c" sub-system defines the user text messages associated with an error ID number. LEAP's errors are classified into event errors, messages, and syntax error. Every sub-system needs to represent the accrued errors through this sub-system. It is expected that this sub-system will be called by every other sub-systems.

Parser "parser.c" This sub-system is mainly used to parse the user command line strings. This parser is similar in work to any parse tree implementation. For example, "The execution phase moves through the tree, up in each level of the parse tree with a relation. Each level's result relation is used as input to the next level, until the root is reached. The relation produced at this level is the result relation for the command"[5].
P_stack.c is the technique used for data parse tree. Parser stack operations and routines make sure that there is no data missing or data redundancy in the relations. LEAP's stack stores the parse tree pointers. The stack is activated when a processing of a LEAP parse tree is required.

Expression Evaluation "cond.c" sub-system provides the functions, which are built to evaluate an expression. LEAP's uses different data types such as, string, integer and Boolean. On the other hand, conditional relational operators are also used.
Relational Language locates a database for modifications such as, insert new element, remove existed element

Main Database sub-system, this sub-system consists of LEAP's relational language "rtional.c", actual database "database.c", database operations "dbase.c", and database relations "relation.c". All database operations are defined and activated from this sub-system. This sub-system together is similar to structure query language. This sub-system is also considered as the core of LEAP because it implements the relational algebra language. This sub-system supports a number of operations on the LEAP database. Get keys, change database, and other database functions. Database functions basically locate cretin key or locate a database for modifications such as, insert new element, remove existed element or modify existed element.

LEAP's uses hashing techniques for data storage "hashing.c". Hashing operations and routines makes sure that there is no data missing or data redundancy in the relations. Any

new hashing value is compared for unique, in case of redundancy a new values or hashing procedures are used.

The attributes sub-system “attributes.c” is responsible for providing the attribute structure of a tuple and link to the stored descriptive information of the tuple.

Utility sub-system is the sub-system, which has all the modules or the functionalities, which work independently from others, usually the routines that interface with the operating system platform. Any operation wants to do something with the OS must pass through this module.

Within figure 2, a summary of results generated from Rigi to all system components is presented and graphed independently. Because the generated graph of Rigi was not clear, so I have evaluated the result and draw it myself. The arrows color dose not mean anything but differentiating the different module interactions.

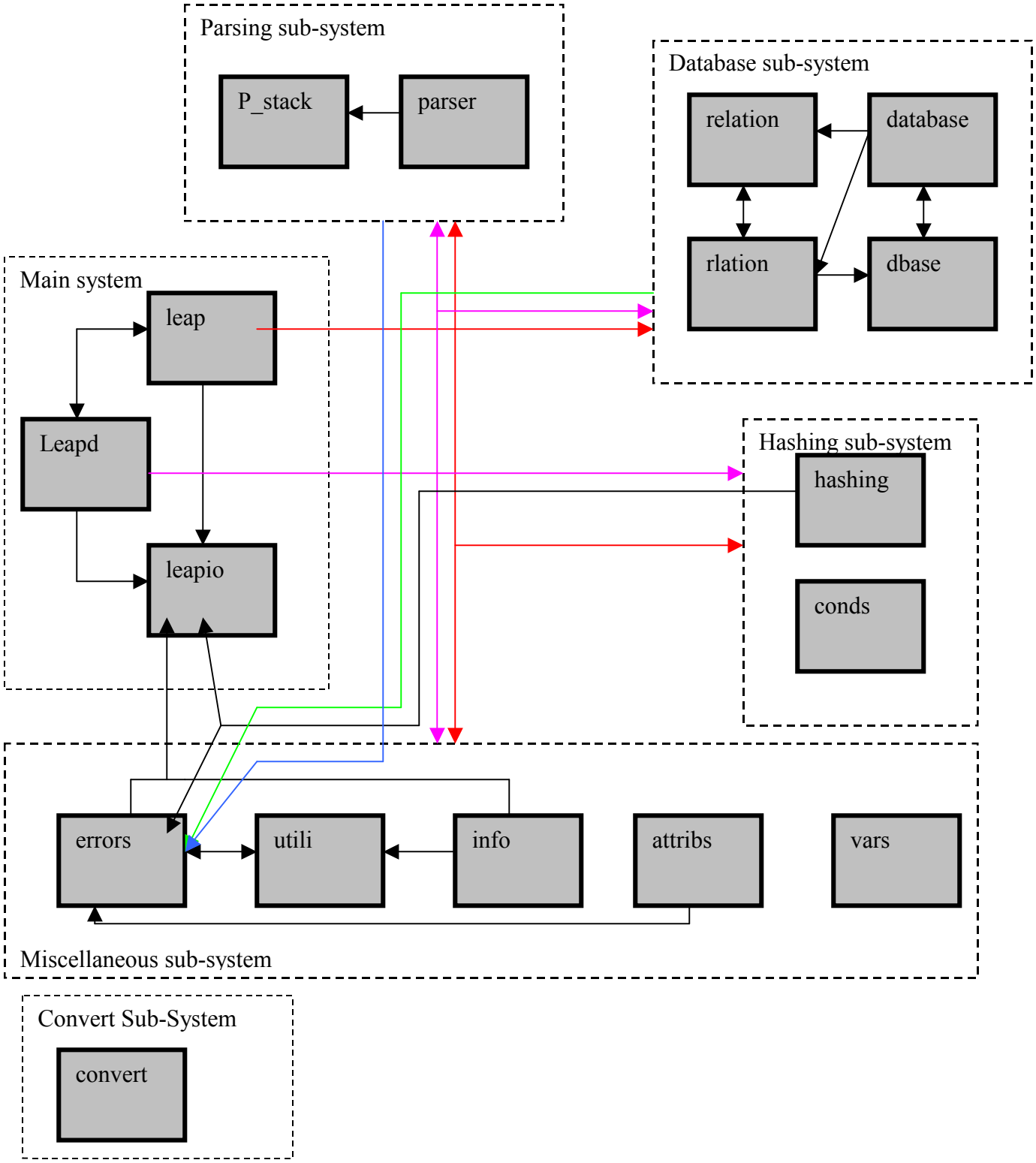


Fig. 2 LEAP's Sub-System Interactions.

4 LEAP Sub-systems Interaction Scenarios

This section shows several internal function-to function calls within the sub-system and the interactions with other sub-systems.

Leap.c is the start point of leaps DBMS. Leap starts by calling the main function, main function is first responsible for initializing the global variables listed in vars.c. then start initializing the error messages and connect them with their ID's, the initialization takes place in errors.c. the most important user interface sub-system is leapio.c. leapio.c start initializing the user interface and calls LEAP's data information from info.c, info.c returns back the copy right and welcome messages

"LEAP 1.2.5 - An extensible and free RDBMS Copyright (C) 1997-1998 Richard Leyton.
LEAP comes with ABSOLUTELY NO WARRANTY; for details type "warranty".
This is free software, and you are welcome to redistribute it under certain conditions; type "copying" for details."

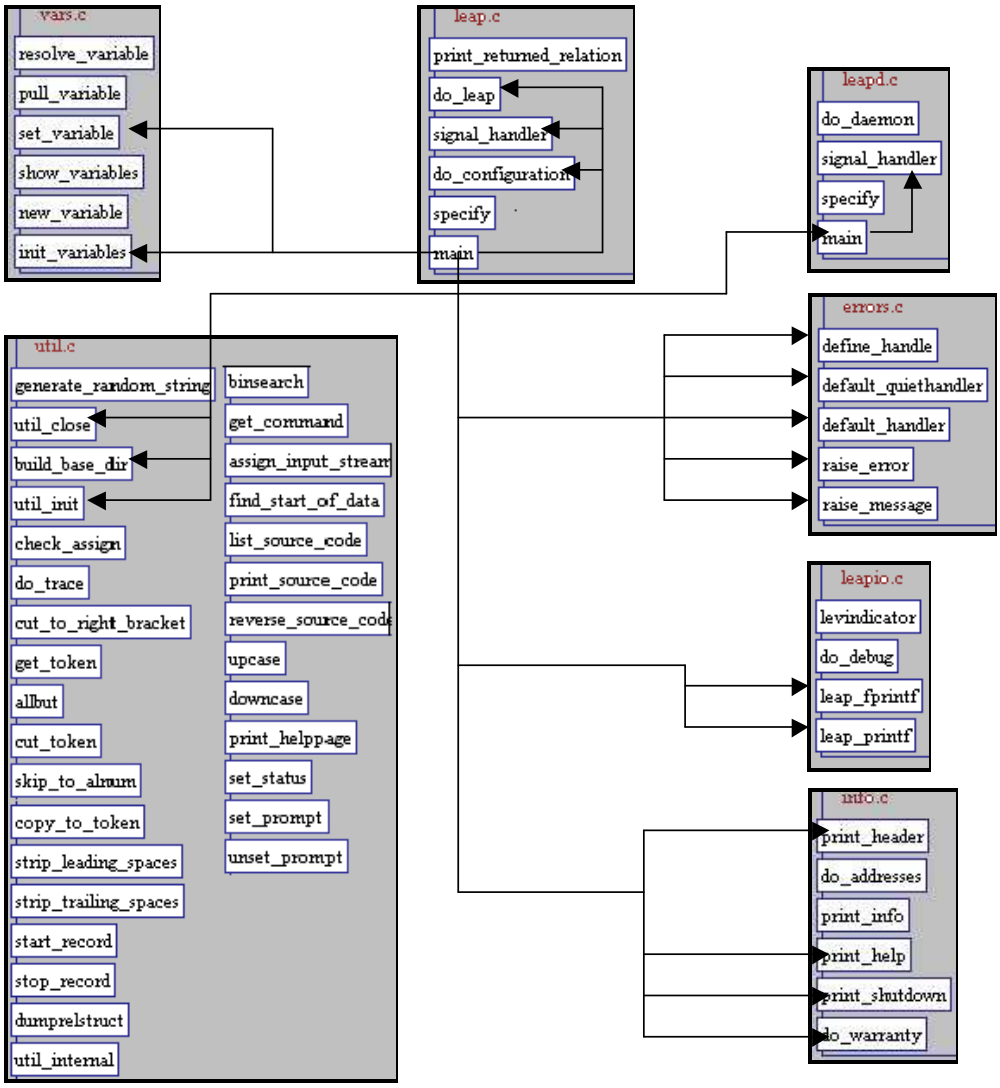
Main also initializing the directory specification and specify the debugging level. Then Leap is now ready to call the database functions, these functions are called from Signal_handler-to-fuctions function, Signal_handler-to-fuctions as shown in figure 4 calls the database functions to open both a temporary and user databases. Main now is ready to call the parser functions and initialize the hashing function. Finally the command line interface starts and the user is ready to enter his/her commands. The following is an output message list generated from the main activation process.

Message: No variables set!
Message: Directory specified [./] not valid. Trying [../]
Message: Debug level set to: 0
Message: Minimum debug level set to: 0
Message: Variables are now set.
Message: Applying command line options...
Message: Completed application of command line options...
Message: LEAP is starting...
Message: Opening [tempdb] database...
Message: Creating hash table for [zzfxrh].
Message: Opening [master] database...
Message: Opening [user] database...
Message: Startup sequence initiated.
Message: Sourcing startup.src in master
Message: Readline library available for command history/editing

Message: Sourcing open.src in user

[user] :-)

Figure 3 and 4 illustrates the main-to-functions initialization calls and Signal_handler-to-fuctions calls. Both of these figures show the complete process of



initialization.

Fig. 3 main-to-function initialization

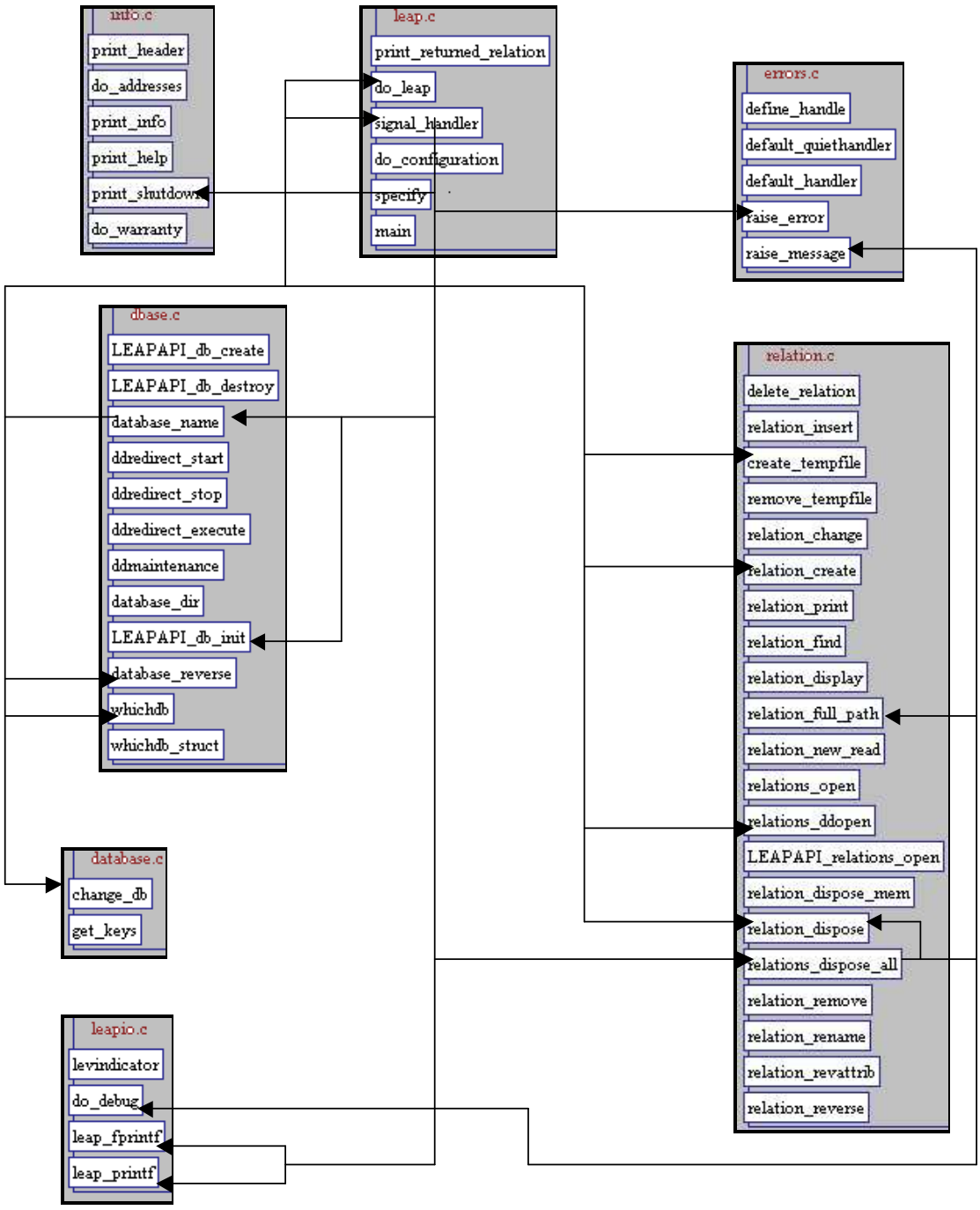


Fig. 4 Signal_handler-to-fuctions calls

5 Data Dictionary

DBMS	: Database Management System
RDBMS	: Relational Database Management System
Relation	: It represents data as being stored in tables
Tuple	: A raw in the relation
Attribute	: a Column in a relation.
SQL	: Structured Query Language
Hashing	: Technique that provides a direct access to records.
Stacks	: List, in which all operations are performed from one end.
Parser	: Process of identifying the grammatical structure of an input.
Parse Tree	: The data structure which a string can be represented in a pictorial form.
Query Lang.	: Language used by DBMS to perform operations on data.

6 References

1. LEAP's Main page http://leap.sourceforge.net/

2. Information Page http://www.dogbert.demon.co.uk/info.htm#newfeatures

3. Publications http://www.leyton.org/publications.html

4. LEAP's development analysis
http://www.dogbert.demon.co.uk/development/notes.htm

5. LEAP's User Guide http://leap.sourceforge.net/user.htm

6. Rigi Reverse engineering toolkit. http://www.rigi.csc.uvic.ca/

7. Most of the Independent module functionality description in section 3.3 has been taken from the source code documentation.