

Source Code Concept Location With Multilingual Support

Nuno Carvalho¹ Alberto Simões² José João Almeida¹

¹Departamento de Informática, Universidade do Minho
`{narcarvalho, jj}@di.uminho.pt`

²Centro de Estudos Humanísticos, Universidade do Minho
`ambs@ilch.uminho.pt`

Per-Fide Workshop
November 22nd, 2011



- today, everything depends on information systems
- humans need to teach computers how to perform tasks
- humans write programs
- programs are written using programming languages
- programs require care
 - have bugs, add features, improvement, ...
 - complex, expensive, time consuming task
- humans devise ways to quicker and better understand programs
- reverse engineering of software – Program Comprehension
- concept location is a common task
 - identify parts of the programs responsible for implementing real life concepts



"reserve an airline ticket"



MIND THE GAP

```
if (seat = request(flight)) &&  
available(seat)  
then  
    reserve(seat, customer)
```





creation

```
import java.awt.*;
import java.util.*;
import java.util.List;
interface State
{
    public State get_initial();
    public State get_goal();
    public Stack get_goals(); // Only if problem has no
    public void add(State s); // solution. throws Exception.
    public boolean equals(State s);
    public void equals(State s); // Overload operator.
    public void equals(State s); // Overload operator.
    public Stack possible_operations();
}
```

```

    // Methods for training
    void State.setInitialValue( ... ); // Returns previous
    public Object previousObject(); // previous state

    // Recursive (n) and Cost(g) functions for this state
    public double g(...); // Estimated cost from start to this
    public double n(...); // Estimated cost to start state to this
    public double F(...); // F() should just return G()

    // If implemented and used, then these two methods could
    // be used to implement a search algorithm
    public void cm_closed(List<...>); // Returns true
    public void cm_open(List<...>); // Returns false
    public void cm_reopen(List<...>); // Returns true

```

```

public int hash_(value);
    // Useful if hash table
import java.awt.*;
import java.util.*;

interface State

public static GetInitial();
public static Get_goal();
public static Get_copy(); // Only if problems has will
public static Get_copy(); throws Exception; // Deep
public static Get_copy(); throws Exception; // Shallow
public void apply(Opator operator); throws Exception;
public void possible(Opator opator); // Only if we want vis
public void print();

BEGINNING_OF_A_VERTEX() { that is methods needed by
// Methods for tracing back along a path (from
// the current state to the initial state)
public Opator previousOpator();
// Returns name of
// (heuristic) dist and Govt (g) functions for this state
public double getHeuristic();
public void setHeuristic(double heuristic);
// Set state start to this
public double getGovt();
public void setGovt(double govt);
// Set state end to this
public double getDist();
public void setDist(double dist);
// Set state to "nearest"
// If no moves are made since last update
public boolean can_closeList();
// Returns true
// if the list can be closed
public void on_closeList(boolean yes_no);
// If yes_no = true, then
public void on_open_list(boolean yes_no);
// If yes_no = true, then
public void hash_(value);
    // Useful if hash table

```

client

date

```
import java.util.*;

interface State

public State get_initial();
public State get_goal();
public Stack get_copy(); // Only if problem has subproblems
public State get_copy(State s); // Only if problem has subproblems
public void apply(Object operator) throws Exception;
public boolean legal_state();
public Stack possible_operators(); // Only if we want visible operators
```

```
beginning of <return> (that is, methods needed  
public State previousState(); // Returns previous  
// state of the system.  
//  
// Distructive (d) and Costly (c) functions:  
// These functions change the system state.  
public double d1(); // Causal lowest bound cost  
// of traversal of the system.  
public double d2(); // Estimated cost of traversal  
// of the system.  
public double d3(); // C(d3) should return sum  
  
// If implemented and used correctly, these five methods could  
// be useful for certain applications.  
public void c1();  
public void c2();  
public void c3();  
public void c4();  
public void c5();  
  
public double h(); // Useful if both methods  
  
creation  
-----  
// Implemented and used correctly, these five methods could  
// be useful for certain applications.  
public boolean o1_closet(List<Object> list);  
public boolean o2_closet(List<Object> list);  
public boolean o3_closet(List<Object> list);  
public boolean o4_closet(List<Object> list);  
public boolean o5_closet(List<Object> list);
```

```
public void onOpenList(boolean ver_no); // Tell  
public int hash_value(); // Useful if hash table  
  
private java.awt.*;  
private java.util.*;  
  
interface State  
  
    public State get_initial();  
    public State get_final();  
    public Stack get_goals(); // Only if problem has goals  
    public State get_copy(); // Returns a copy of this Exception.  
    public void apply(State m);  
    public void apply(Object state) throws Exception;  
    public boolean legal();  
    public void print();
```

client

```
port java.awt.*;
import java.util.*;
interface State

public State get_initial();
public State get_goal(); // Only if problem has wait states
public void solve() throws Exception; // Deep search
public void solve(State s);
public void solve(State s, State operator) throws Exception; // Shallow search
public void print_state();
public void paint(Graphics g); // Only if we want visualization
```

```

    * Methods for tracking will also print a path through
      the state space preconditions.  Returns previous
      state if no solution found or None if no solution
      exists.

  /> Heuristic (h) and Cost (f) functions
  public double h(...); // Current lowest known cost
  public double NO(); // Estimated cost of traversal
  public double TD(); // f() should return it()

  II) implemented and used these methods could
  be useful in some cases:
  public boolean can_open(List<Node>...); // Returns true if
  public void on_closed(List<Node>...); // Node was closed
  public void on_expanded(List<Node>...); // Node was expanded
  public void on_hl_reach(); // Useful if heap table
  public void on_hl_update(); // Useful if heap table

  public State on_error(...); // throws Exception
  public void on_error(String operator); // throws Exception
  public Track possible_operators();
  
```

creation

creation * Methods for tracking will also print a path through
the state space preconditions. Returns previous
state if no solution found or None if no solution
exists.

creation

creation

```

    public void previousState() { // Returns previous
        Object previousObject = previousObject(); // previous state
        previousObject.previousObject(); // previous state
    }

    // Bureaucrat (a) and Costie (functions for this state
    public void enter() { // Enter state to this
        stateEntered++; // Enter state to this
        if (stateEntered > 1) {
            System.out.println("Error: too many entries");
            return;
        }
        if (stateEntered == 1) {
            System.out.println("First entry to state");
        }
    }

    public double add(double a, double b) { // Add two numbers
        double sum = a + b;
        System.out.println("Sum = " + sum);
        return sum;
    }

    public double divide(double a, double b) { // Divide two numbers
        double result = a / b;
        System.out.println("Result = " + result);
        return result;
    }

    // If implemented and used, then these five methods could
    public boolean canCloseList(boolean val) { // Returns true
        if (val) {
            System.out.println("Can close list");
        } else {
            System.out.println("Cannot close list");
        }
        return val;
    }

    public void canOpenList(boolean val, int no) { // To
        if (val) {
            System.out.println("Can open list " + no);
        } else {
            System.out.println("Cannot open list " + no);
        }
    }

    public void canWriteList(boolean val, int no) { // To
        if (val) {
            System.out.println("Can write list " + no);
        } else {
            System.out.println("Cannot write list " + no);
        }
    }

    public boolean canWriteTable(boolean val) { // Default if both tables
        if (val) {
            System.out.println("Can write table");
        } else {
            System.out.println("Cannot write table");
        }
        return val;
    }
}

```

creation

```

public Stack<String> operators()); // Only if we want plus
public void main(Scanner s); // Only if we want plus

BEGINNING of a METHOD (that is needed to
    Methods for tracking back along a path [method name]
    previousState()); // Returns previous
    previousOperator()); // Returns operator
    previousObject()); // Returns object

Heuristic (h) and Cost(g) functions for this state
    heuristic((S, G)) = CASHIER.JOHNS_KNIGHT_COST;
    cost((S, G)) = 0;

public double h(); // From start state to this
    state // From start state to "mean"
public double g(); // ((t)) should just return g()

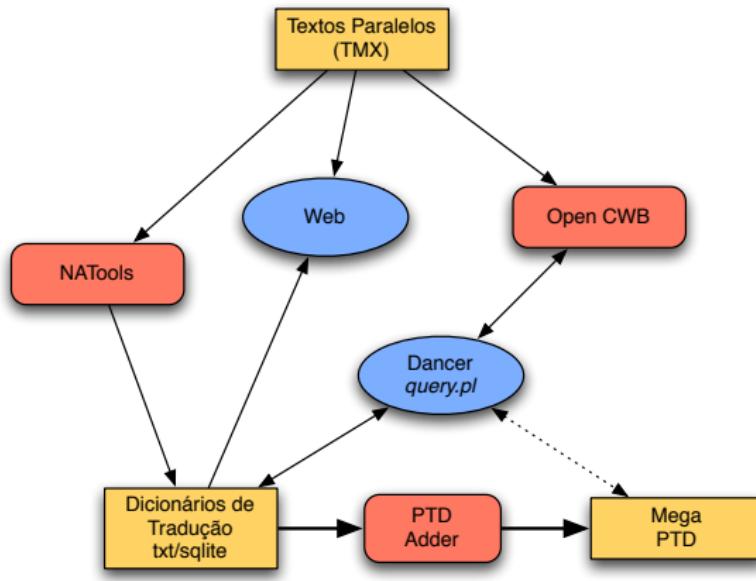
```

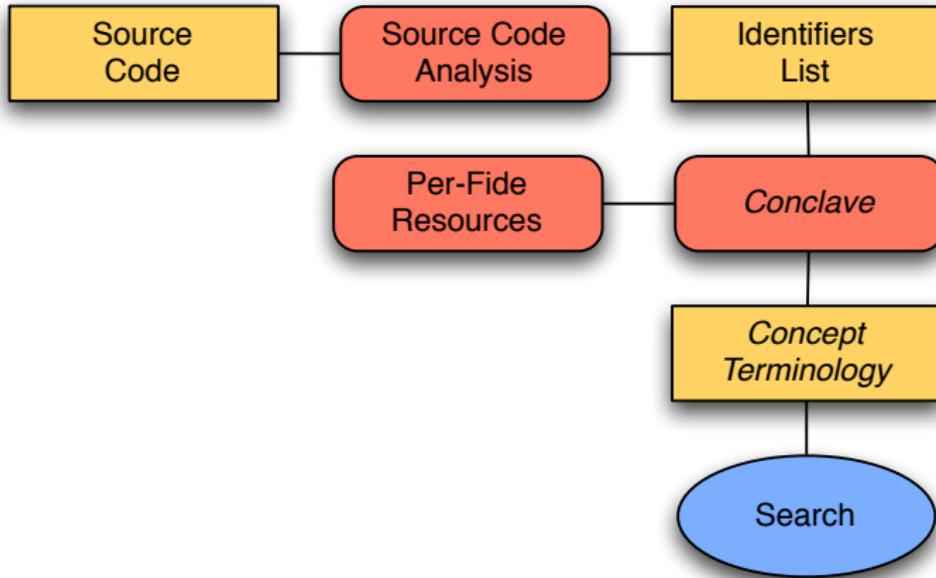
```
public boolean ca.close(list); // Returns true if list was closed
public void ca.open(list); // Returns false if list was already open
public void ca.open(list boolean yes_no); // Tells ca to open list
public void ca.open(list boolean yes_no); // Tells ca to open list
public int hash_val(); // Useful if hash table
public int hash_val(boolean); // Useful if hash table

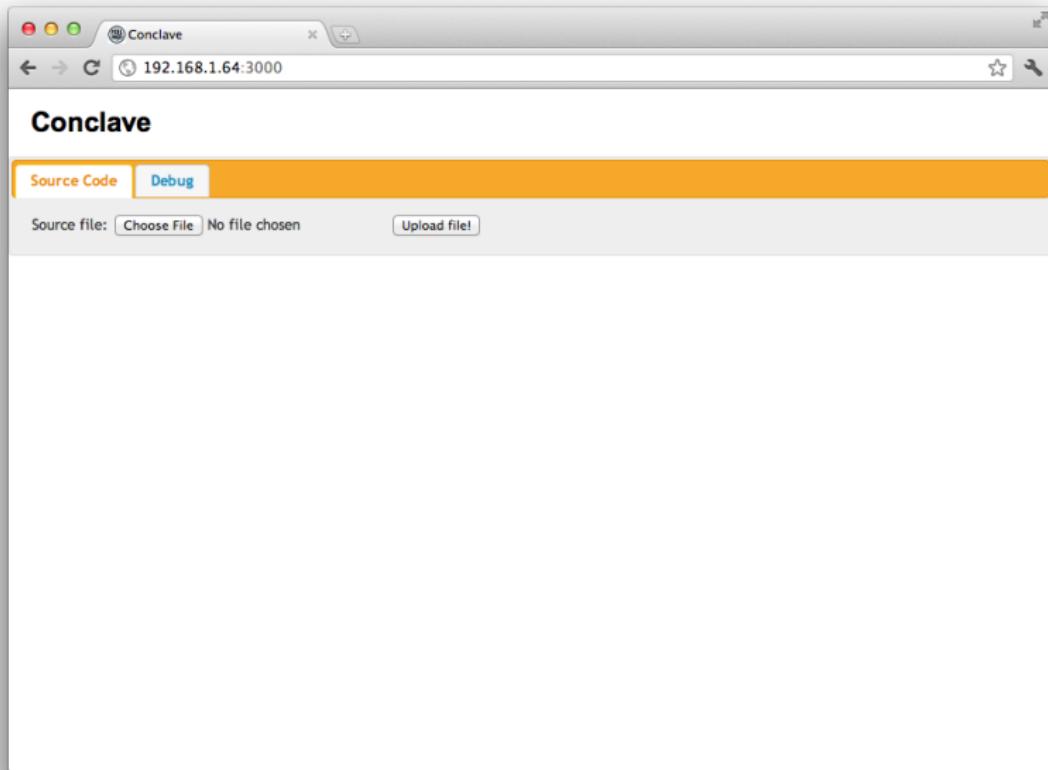
java.awt.*;
java.awt.*;
java.awt.*;

class State {
    State get_initial();
    void set_goal();
    void set_goal(boolean); // Only if problem has wait states
    void get_prob(); // Returns exception. If problem has wait states
    void apply(Object problem) throws Exception; // Returns exception. If problem has wait states
    void paint(Graphics); // Only if we want to see state
    void check(); // Returns true for traces back along a plan (path)
    void print_plan(); // Returns previous state
    void print_prob(); // Returns problem state
    Heuristic (H and Cost) functions
    void print_prob(); // Returns previous state
    void print_prob(); // Returns previous state to this
    public double A(); // Estimated cost of travel from start state to this
    public double f(); // Should just return g()
    public double f(); // Should just return g()
}

public class Problem {
    public boolean ca.close(list); // Five methods could be used here
    public boolean ca.open(list); // Returns false if list was already open
    public boolean ca.open(list boolean yes_no); // Returns true if list was closed
    public void ca.open(list boolean yes_no); // Tells ca to open list
    public void ca.open(list boolean yes_no); // Tells ca to open list
    public int hash_val(); // Useful if hash table
    public int hash_val(boolean); // Useful if hash table
}
```









The screenshot shows a web browser window titled "Conclave" at the URL "192.168.1.64:3000". The browser has standard navigation buttons (back, forward, search) and a toolbar with tabs: "Source Code" (selected), "PiDL", "Program Identifiers", "Concept Location", and "Debug". Below the toolbar, there is a file input field labeled "Source file:" with the placeholder "Choose File" and a message "No file chosen". To the right of the input field is a button "Upload file!". A large text area contains the following C-like pseudocode:

```
int username;
int password;

int init() {
    set_defaults();
    db_init();

    return 1;
}

int authenticate() {
    return validate(username, password);
}

int main() {
    init();

    return 1;
}
```

At the bottom of the text area is a button "Update source code". The entire interface is contained within a light gray box with a blue border.



The screenshot shows a web browser window titled "Conclave" at the URL "192.168.1.64:3000". The browser has standard navigation buttons (back, forward, search) and a star icon. The main content area is titled "Conclave" and contains a table of program identifiers.

Constants		
Name	—	Filename

Variables		
Name	Proc	Filename
password	GLOBAL	/tmp/conclave/t1.c:3
username	GLOBAL	/tmp/conclave/t1.c:2

Procedures		
Name	—	Filename
authenticate	—	/tmp/conclave/t1.c:14
main	—	/tmp/conclave/t1.c:20
init	—	/tmp/conclave/t1.c:10



The screenshot shows a web browser window titled "Conclave" at the URL "192.168.1.64:3000". The page has a header with five tabs: "Source Code", "PiDL", "Program Identifiers", "Concept Location" (which is highlighted in orange), and "Debug". Below the tabs is a search form with a text input field containing "<term>" and a "Locate" button. A descriptive message "Use the form above to search for a term.." is displayed below the form. The main content area is currently empty.



The screenshot shows a web-based IDE interface titled "Conclave". The address bar indicates the URL is 192.168.1.64:3000. The main window title is "Conclave". Below the title, there is a navigation bar with tabs: "Source Code" (which is highlighted in orange), "PiDL", "Program Identifiers", "Concept Location", and "Debug". A search bar contains the text "username" and a "Locate" button.

```
int username;
int password;

int init() {
    set_defaults();
    db_init();

    return 1;
}

int authenticate() {
    return validate(username, password);
}

int main() {
    init();

    return 1;
}
```



The screenshot shows the Conclave IDE interface. The title bar says "Conclave" and the address bar shows "192.168.1.64:3000". The main window has tabs: "Source Code" (selected), "PiDL", "Program Identifiers", "Concept Location", and "Debug". A search bar contains "utilizador" and a "Locate" button. On the left is a code editor with the following C-like pseudocode:

```
int username;
int password;

int init() {
    set_defaults();
    db_init();

    return 1;
}

int authenticate() {
    return validate(username, password);
}

int main() {
    init();

    return 1;
}
```

On the right, a tooltip box highlights the word "username" and lists suggestions: "utilizador", "choose", "user", "set", "nome", "name", "username", and "now".



The screenshot shows a web-based IDE interface titled "Conclave" at the URL 192.168.1.64:3000. The interface has a header with tabs: Source Code, PiDL, Program Identifiers, Concept Location, and Debug. The "Concept Location" tab is currently selected. Below the tabs is a search bar with the word "init" and a "Locate" button. The main content area displays a code editor with the following C-like pseudocode:

```
int username;
int password;

int init() {
    set_defaults();
    db_init();

    return 1;
}

int authenticate() {
    return validate(username, password);
}

int main() {
    init();
    return 1;
}
```

The word "init" is highlighted in yellow across all three occurrences in the code.



Match Another Term, Same Concept

The screenshot shows the Conclave IDE interface. The title bar says "Conclave" and the address bar shows "192.168.1.64:3000". The menu bar includes "Source Code", "PIDL", "Program Identifiers", "Concept Location" (which is highlighted in orange), and "Debug". A search bar contains the text "initialization" and a "Locate" button. Below the search bar is a code editor window displaying C-like pseudocode:

```
int username;
int password;

int init() {
    set_defaults();
    db.init();
    return 1;
}

int authenticate() {
    return validate(username, password);
}

int main() {
    init();
    return 1;
}
```

To the right of the code editor is a vertical list of code completion suggestions. The first few items are "init", "the", "establishing", "connection", and "connecting". Further down the list are "rest", "of", "not", "initialize", "init", "to", "inicializar", "making", "makes", "at", "iniciar", "startup", "graphical", "easy", "open", "gui", "starting", "libbonoboui", "Initialization", "manage", and "start". The suggestion "Initialization" is highlighted with an orange border.





- share Per-Fide resources
 - these can be used in ways we don't think of (people have an incredible imagination)
 - others can take advantage of the projects' effort
- use of NLP resources in Program Comprehension
- clear advantages
 - more than multilingual support
 - more trust in translations, specialized corpora
 - look also for similar terms
 - (more)
- future work
 - enrich concepts terminology (based on comments for example)
 - more robust tool, interface improvements, ...
 - elegant way to share resources

Thank You!