Refactoring with Wrangler, updated

Huiqing Li, Simon Thompson
Computing Lab, University of Kent

György Orosz and Melinda Tóth
Eötvös Lórand University, Budapest

www.cs.kent.ac.uk/projects/refactor-fp/
Outline

• Wrangler – an Erlang refactoring tool.
• Recent Developments:
  – Improved infrastructure
  – Improved usability
  – Data-related refactorings
  – Process-related refactorings
  – Integration with Eclipse
Wrangler

• Wrangler – an Erlang Refactoring tool embedded in Emacs and Eclipse.
• AAST-based program analysis and transformation.
• Work with multiple modules.
• Support undo of refactorings.
• Preserve both layout and comments as much as possible.
• Designed to be used in practice.
-module(test).
-import(refac_util,[get_range/4, do_add_range/3, extend_backwards/3, extend_forwards/3]).
-export(add_range/2).

add_range(FName, AST) ->
    {ok, Toks} = refac쨉e(FName, AST, refac_util:full_buTP(f), refac_util:full_buTP(f)),
    do_add_range(Node, Toks).

do_add_range(Node, Toks) ->
    {L, C} = refac_syntax:case_expr(Node),
    case refac_syntax:type(Node) of
        case_expr ->
            A = refac_syntax:case_expr_case_expr (Node),
            Lc = glast("refac_util:do_add_range,case_expr", C),
            {S1, _E1} = get_range(A),
            {S2, E2} = get_range(Lc),
            S11 = extend_forwards(Toks, S1, 'case'),
            E21 = extend_forwards(Toks, E2, 'end'),
            refac_syntax:add_ann({range, [{S11, E21}]}, Toks),
            cond_expr ->
                Cs = refac_syntax:cond_expr_clauses(Node),
                Hd = glast("refac_util:do_add_range,cond_expr", C),
                La = glast("refac_util:do_add_range,cond_expr", C),
                {S1, _E1} = get_range(Hd),
                {S2, E2} = get_range(La),
                S11 = extend_forwards(Toks, S1, 'cond'),
                E21 = extend_forwards(Toks, E2, 'end'),
                refac_syntax:add_ann({range, [{S11, E21}]}, Toks),
            end.
end.
Recent Developments
-- Improved Infrastructure

• The original Wrangler workflow.
Recent Developments
-- Improved Infrastructure

• The current Wrangler workflow.

Diagram:
- Wrangler App
- Wrangler Sup
- AST server
- callgraph server
- modulegraph server
- undo server
- error logger
- Program source
- refactorer
- program rendering
Recent Developments
-- Improved User Experience

• Program appearance preservation.
  • Only pretty-print functions/attributes that have been changed by the refactoring process.
  • A function’s original layout is taken into account when it is pretty-printed.

• Refactoring code with syntax errors.
  • Functions with syntax errors are not affected by the refactoring process.

• Efficiency Enhancement.
Recent Developments
-- Data-related Refactorings

• Tuple function parameters -- group a number of consecutive arguments of a function into a tuple.

-module(tup1).
-export([f/3]).
f(A, B, C) -> A+B+C.
-module(tup2).
-export([g/3]).
g(X, Y, Z) -> tup1:f(X, Y, Z).

-module(tup1).
-export([f/2]).
f({A, B}, C) -> A+B+C.
-module(tup2).
-export([g/3]).
g(X, Y, Z) -> tup1:f({X, Y},Z).
Recent Developments
-- Data-related Refactorings

• From Tuple to Record -- turn a tuple function parameter to a record.

-module(record).
-export([g/1]).
f({A, B}, C) -> A+B+C.
g(X) -> f({X, 2*X},3*X).

-module(record).
-export([g/1]).
-record(rec,[first,second]).
f(#rec{first=A, second=B},C) -> A+B+C.
g(X) -> f(#rec{first=X,second=2*X},3*X).
Recent Developments
-- Process-related Refactorings

• Some process-related bad code smells.
  • Code for a single process spans multiple modules, or code for more than one kind of process exists in the same module.
  • Process and message passing, or sequential function call?
  • Name of a registered process does not reflect its role or functionality.
  • Send/receive untagged messages.
  • Register a process that only lives a short time, or not register a process that lives a long time.
  • Generic OTP behaviours not used when doing so is more appropriate.
Recent Developments
-- Process-related Refactorings

• Some process-related refactorings we have studied.
  • Register a process.
  • From function to process.
  • Rename a registered process.
  • Add a tag to the messages received by a server process.
Recent Developments
-- Process-related Refactorings

Demo
Recent Developments
-- Process-related Refactorings

• Major challenges.
  • Processes in an Erlang program are syntactically implicit.
  • Unlike a function or module, a process in Erlang does not have a clear syntactically specified body or scope.
  • Implicit connection between a process identifier and the process identified.
  • Process context-dependent evaluation, e.g. self().
Recent Developments
-- Process-related Refactorings

• Example refactoring: register a process.
  • Side conditions:
    • The process name should not have been used as a process name in the program.
    • The selected process should not have been registered.
    • Should multiple instances of the process exist during run time, they should not co-exist at the same time.
  • Transformation rules:
    • Add the registration expression;
    • Replace the process identifier in a send expression when doing this is safe.
Recent Developments
-- Process-related Refactorings

• Approaches we have taken to support process-oriented refactoring.

• Extend the existing AAST representation with process information using static data flow analysis; Given an Erlang program, we would like to know whether a variable represents a PID, and if yes, which spawn expression(s) this PID is associated with.

```erlang
f() -> Pid = spawn(test, fun1, [a]),   g(Pid),
       Pid1 = spawn(test, fun1, [b]),   g(Pid1).
       g(Pid2) -> Pid2 ! message.
       h(Pid3) -> Pid3 ! another_message.
```

**Pid** :  `{pid, [{initial_fun, {test, fun1, 1}},  {spawn, {test, f, 0, 1}}]}`

**Pid1** : `{pid, [{initial_fun, {test, fun1, 1}},  {spawn, {test, f, 0, 2}}]}`

**Pid2** : `{pid, [{initial_fun, {test, fun1, 1}},  {spawn, {test, f, 0, 1}}],
              [{initial_fun, {test, fun1, 1}},  {spawn, {test, f, 0, 2}}]}`

**Pid3** : `{pid, [any]}`
Recent Developments
-- Process-related Refactorings

• Approaches we have taken to support process-oriented refactoring. (cont)
  • The use of slicing techniques to reduce the scope of analysis and the number of uncertain cases;
    • Backward slicing
    • Forward slicing
  • The use of trace information.
Recent Developments
-- Process-related Refactorings

• Approaches we have taken to support process-oriented refactoring. (cont)

• Backward slicing could also help evaluation.

```erlang
start() -> Prefix = "ch1", State = [1,2]
    start(Prefix, State).
start(Prefix, State) ->
    ProcessName = list_to_atom(Prefix++"_proc"),
    Pid = spawn(ch1, init,[ProcessName, State]),
    register(ProcessName, Pid).

fun(Prefix) ->
    ProcessName = list_to_atom(Prefix++"_proc"),
    ProcessName
end (begin Prefix = "ch1", Prefix end).
```
Recent Developments
-- Eclipse Integration

• Motivation.

  • There are some limitations to the way in which Wrangler is integrated into the Emacs editor.
    • the concept of project;
    • no connection between the refactorer and the build infrastructure;
    • separate notions of ‘undo’ for editing and refactoring.

• The refactoring support and API in Eclipse.

• Some Erlang users use Eclipse + Erlide as their as their Erlang IDE.
Recent Developments
-- Eclipse Integration

• Strategy
  • Make as few changes to Wrangler as necessary.
  • Make use of the refactoring API of Eclipse, which supports the integration into various aspects of the infrastructure of Eclipse.
    • the ‘refactor’ menu
    • refactoring previews
    • ‘undo’ and ‘redo’ support
Recent Developments
-- Eclipse Integration

• Integration Challenges.
  • The workflows followed by the refactoring API from Eclipse and Wrangler are not quite the same.

Eclipse refactoring API workflow
1. user initiates the refactoring
2. initial check of preconditions
3. user interaction
4. pre-condition checking
5. transformation
6. preview of new program
7. apply changes

The Wrangler Workflow
1. user initiates the refactoring
2. user interaction
3. pre-condition checking
4. transformation
5. apply changes
Recent Developments
-- Eclipse Integration

• Integration Challenges. (cont.)
  • The refactoring API does not allow user interaction during the refactoring process, while Wrangler could have very rich interactions with the user for some refactorings.
  • Renaming of module names proved to be difficult within the refactoring API framework.
  • Some modifications had to be made to Wrangler to better support the integration.
Conclusions and Future work

- The infrastructure of Wrangler has been improved to support better user experience.
- More advanced refactorings have been studied.
- Integration of Wrangler into Eclipse is possible, though not trivial.
- In the future, we would like to further investigate the use of trace information to help the refactoring process, especially process-oriented refactorings.
- To continue the integration with Eclipse, in collaboration with the Erlide team.
- Wrangler 0.5 released this week.
Thank you