Technology preview

Inheritance in Erlang

Basic module inheritance

```
%% Simple example

-module(new_module).

-extends(old_module).

-export([g/1]).

g(X) ->
    ...
```

- 'extends' declaration enables inheritance.
- All function calls to new_module:f(...) will be redirected to old_module:f(...) if f is not exported from new_module.
- Overriding follows naturally.

The interesting part: inheritance and parameterized modules

```
%% Basic parameterized
%% module example

-module(alpha, [X,Y]).

-export([a/0,b/1,c/2]).

a() ->
 {?MODULE,a,[X,Y],[]}

b(S) ->
 {?MODULE,b,[X,Y],[S]}

c(S,T) ->
 {?MODULE,c,[X,Y],[S,T]}
```

- Automatic new/2 function generated.
- M = alpha:new(x, y)
- M:a() = {alpha, a, [x,y], []}
- M:b(1) = {alpha, b, [x,y], [1]}
- M:c(1,2) = {alpha, c, [x,y], [1,2]}

Extending a parameterized module

```
%% Parameterized
%% module using
%% inheritance

-module(beta, [X,Y]).
-extends(alpha).

-export([a/0,b/1]).

a() ->
 {?MODULE,?BASE_MODULE,
  a,[X,Y],[]}

b(S) ->
 {?MODULE,?BASE_MODULE,
  b,[X,Y],[S]}
```

- generated new/2 will call alpha:new/2.
- M = beta:new(x, y)
- M:a() = {beta, alpha, a, [x,y], []}
- M:b(1) = {beta, alpha, b, [x,y], [1]}
- M:c(1,2) = {alpha, c, [x,y], [1,2]}

User-defined new/N functions

```
%% Controlling the
%% construction

-module(gamma, [X,Y]).
-extends(beta).

-export([a/0,new/1]).

new(Z) ->
 B = ?BASE_MODULE:new(Z,
  17),
 instance(B,Z,42).

a() ->
 {?MODULE,?BASE_MODULE,
  a,[X,Y],[]}
```

- No new/N function is generated if the user defines one or more new/K functions.
- M = gamma:new(z)
- M:a() = {gamma, beta, a, [z,42], []}
- M:b(1) = {beta, alpha, b, [z,17], [1]}
- M:c(1,2) = {alpha, c, [z,17], [1,2]}

instance/N: the real constructor

- Always generated for parameterized modules
- Returns an instance of the current module
- Takes an initial Base parameter if the module contains an extends-declaration.
- Not exported.
- Should only be called from new/K functions.
- Generated new/N calls instance/N.
- Parameters to new/K functions do not have to be directly mapped to instance/N parameters – but by default, they are.

THIS and BASE

- Functions in parameterized modules can reference the hidden THIS parameter (bound to the running module instance itself).
- Functions in parameterized modules with inheritance get an extra hidden parameter BASE (bound to the instance of the base module).
- Example: f(X) -> {some_tag, BASE:f(X)}.

Now I'm done

Please don't hurt me