Na tym kończy się historia Erlanga. I tu pojawia się pytanie, jeżeli Erlang jest tak wspaniały, to po co ci Elixir. Odpowiedź znajdziesz w następnej części tekstu.

1.2. O Elixirze

Elixir jest językiem alternatywnym dla wirtualnej maszyny Erlanga, który pozwala na pisanie przejrzystszego, bardziej kompaktowego kodu, który odzwierciedla twoje intencje. Swoje programy piszesz w Elixirze i uruchamiasz je na BEAM.

Elixir jest otwartym oprogramowaniem, nad którym pracę rozpoczął José Valim. W odróżnieniu od Erlanga Elixir jest w większym stopniu efektem współpracy; obecnie w jego powstanie wkład ma około 700 osób. Nowe funkcje są regularnie omawiane na listach mailingowych, na GitHub, gdzie są omawiane problemy, oraz na kanale IRC freenode #elixir-lang. Ostateczne zdanie należy do José, ale cały projekt jest faktycznie efektem współpracy nad otwartym oprogramowaniem, co przyciąga ciekawą mieszankę doświadczonych weteranów Erlanga oraz utalentowanych młodych programistów. Kod źródłowy jest dostępny na repozytorium GitHub pod adresem https://github.com/elixir-lang/elixir.

Elixir bazuje na środowisku Erlang. Wynikiem kompilacji kodu źródłowego Elixira są pliki kodu bajtowego zgodnego z BEAM, mogące działać na instancji BEAM – możesz używać bibliotek Erlanga z Elixira, i odwrotnie. Nie ma takiej rzeczy w Erlangu, której nie byłbyś w stanie zrobić w Elixirze, i zazwyczaj kod Elixira jest równie sprawny, jak kod jego odpowiednika w Erlangu.

Elixir jest semantycznie zbliżony do Erlanga: wiele z jego konstrukcji językowych odnosi się bezpośrednio do odpowiednika w Erlangu, jednak posiada on kilka dodatkowych konstrukcji, które zmniejszają duplikację w kodzie oraz wymagają znacznie mniej szablonowych (rutynowych, powtarzalnych) konstrukcji. Ponadto porządkuje niektóre istotne elementy bibliotek, wprowadza lukier składniowy oraz jednolite narzędzie do tworzenia i pakowania rozwiązań. Wszystko, co da się zrobić w Erlangu, można wykonać również w Elixirze, i na odwrót. Z mojego doświadczenia wynika, że rozwiązanie w postaci Elixira jest zazwyczaj łatwiejsze do stworzenia i obsługi.

Przyjrzyjmy się temu, jak Elixir usprawnia pewne funkcje Erlanga. Zacznijmy od zbędnie powtarzalnego kodu.

1.2.1. Uproszczenie kodu

Jedną z najważniejszych zalet Elixira jest jego zdolność do znacznej redukcji zbędnego powtarzania pewnych struktur czy wyrażeń, co przekłada się na uproszczenie kodu, który łatwiej napisać i obsługiwać. Sprawdźmy, co to oznacza, zestawiając ze sobą kod Erlanga i Elixira.

Często stosowanym podstawowym elementem współbieżnych systemów Erlanga jest proces serwera. Procesy serwera można rozumieć jako coś w rodzaju współbieżnych obiektów. Przechowują one stan prywatny i potrafią wchodzić w interakcje z innymi procesami poprzez wiadomości. Będąc współbieżnymi, różne procesy mogą być wykonywane równolegle. Typowe systemy Erlanga polegają w znacznym stopniu na procesach, których mogą być tysiące, a nawet miliony.

Poniższy przykład kodu Erlanga implementuje prosty proces serwera dodający dwie liczby.

Listing 1.1. Proces serwera opartego na Erlangu dodającego dwie liczby

```
-module(sum_server).
-behaviour(gen_server).

-export([
  start/0, sum/3,
  init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2,
  code_change/3
]).

start() -> gen_server:start(?MODULE, [],   []).
sum(Server, A, B) -> gen_server:call(Server, {sum, A, B}).

init(_) -> {ok, undefined}.
handle_call({sum, A, B}, _From, State) -> {reply, A + B, State}.
handle_cast(_Msg, State) -> {noreply, State}.
handle_info(_Info, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_OldVsn, State, _Extra) -> {ok, State}.
```

Nawet bez znajomości Erlanga można dojść do wniosku, że to pokaźnych rozmiarów kod jak na coś, co ma dodać jedynie dwie liczby. Co prawda owo dodanie jest współbieżne, ale niezależnie od tego, z powodu liczby linii kodu, trudno to ocenić. Nie jest też na pierwszy rzut oka oczywiste, do czego ten kod służy. Ponadto niełatwo taki kod napisać. Nawet po latach tworzenia w Erlangu systemów dostępnych na produkcji nadal nie jestem w stanie napisać tego bez zaglądania do dokumentacji czy kopiowania i wklejania z uprzednio napisanego kodu.

Problem Erlanga polega na tym, że tego szablonowego kodu praktycznie nie da się usunąć, nawet jeżeli wygląda identycznie w różnych miejscach (co, jak wynika z mojego doświadczenia, zachodzi w większości przypadków). Język ten nie daje możliwości pozbycia się tego szumu. Istnieje wprawdzie metoda ograniczenia powtarzalnego kodu przez zastosowanie funkcji `parse transform`, ale jest ona niezgrabna i trudna w zastosowaniu. W rzeczywistości programiści Erlanga piszą procesy serwera, używając powyższego wzorca.

Procesy serwera są istotnymi i często wykorzystywanymi narzędziami Erlanga, dlatego też programiści tego języka są niestety zmuszeni do pracy z tym stale kopiowanym i wklejanym powtarzalnym kodem. O dziwo, wielu programistów się do tego przyzwyczaja, najprawdopodobniej dzięki możliwościom, jakie potrafi dla nich zdziałać BEAM. Mówi się, że Erlang czyni rzeczy trudne prostymi i komplikuje rzeczy proste. Mimo to powyższy kod dowodzi, że powinna istnieć metoda na to, by zrobić to lepiej.

Spójrzmy na ten sam proces serwera, ale w przypadku Elixira.