OPEN TELECOM PLATFORM

OTP stands for *Open Telecom Platform*, although it's not that much about telecom anymore (it's more about software that has the property of telecom applications, but yeah.) If half of Erlang's greatness comes from its concurrency and distribution and the other half comes from its error handling capabilities, then the OTP framework is the third half of it.
During the previous chapters, we've seen a few examples of common practices on how to write concurrent applications with the languages' built-in facilities: links, monitors, servers, timeouts, trapping exits, etc. There were a few 'gotchas' here and there on the order things need to be done, on how to avoid race conditions or to always remember that a process could die at any time. There was also hot code loading, naming processes and adding supervisors, to name a few.

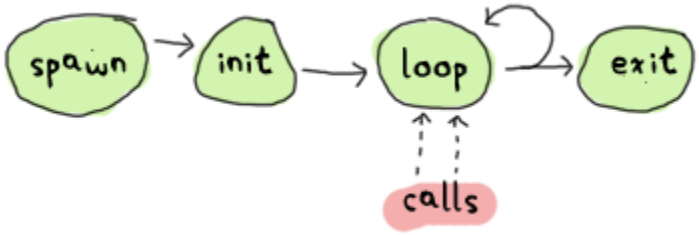
Doing all of this manually is time consuming and sometimes prone to error. There are corner cases to be forgotten about and pits to fall into. The OTP framework takes care of this by grouping these essential practices into a set of libraries that have been carefully engineered and battle-hardened over years. Every Erlang programmer should use them.

The OTP framework is also a set of modules and standards designed to help you build applications. Given most Erlang programmers end up using OTP, most Erlang applications you'll encounter in the wild will tend to follow these standards.

The Common Process, Abstracted

One of the things we've done many times in the previous process examples is divide everything in accordance to very specific tasks. In most processes, we had a function in charge of spawning the new process, a function in charge of giving it its initial values, a main loop, etc.

These parts, as it turns out, are usually present in all concurrent programs you'll write, no matter what the process might be used for.



The engineers and computer scientists behind the OTP framework spotted these patterns and included them in a bunch of common libraries. These libraries are built with code that is equivalent to most of the abstractions we used (like using references to tag messages), with the advantage of being used for years in the field and also being built with far more caution than we were with our implementations. They contain functions to safely spawn and initialize processes, send messages to them in a fault-tolerant manner and many other things. Funnily enough, you should rarely need to use these libraries yourself. The abstractions they contain are so basic and universal that a lot more interesting things were built on top of them. Those libraries are the ones we'll use.

In the following chapters we'll see a few of the common uses of processes and then how they can be abstracted, then made generic. Then for each of these we'll also see the corresponding implementation with the OTP framework's behaviours and how to use each of them.

Source : http://learnyousomeerlang.com/what-is-otp