POE: The Perl Object Environment

Rocco Caputo
<rcaputo@pobox.com>

July 11, 2003

Abstract

POE is a toolkit for rapidly developing networking applications in Perl. POE's powerful multitasking framework lets applications perform several tasks at once. POE's message passing capabilities allow applications to be distributed across a network of machines. POE is mature, robust, and highly considered. POE is free and open, and can be field-customized for an organization's unique needs. POE is well supported by many people, and there exists a number of free extensions on the Comprehensive Perl Archive Network[1].

[1] Often abbreviated as "CPAN" and pronounced as "See-pan".

Contents

1 A Rapid Development Toolkit for Networking

> *There's more than one way to do it.*
>
> — *Larry Wall*

Network applications often follow time-worn programming patterns. POE was designed to implement those patterns once and for all, reducing the time and effort wasted by repeatedly rewriting them.

One size doesn't fit all, though. That's why POE provides three levels of abstraction. Each strikes a different balance between development control and developer effort. The result is a sweet spot for every task, and the ability for a single application to mix and match them.

- High-level networking.

  POE's high-level components enable developers to rapidly create and deploy functional network applications with the least amount of effort.

  With POE, the networking infrastructure for a basic internet service can be set up in about 20 lines of Perl. Developers are then free to focus their efforts on the features that make their applications unique.

- Mid-level networking.

  POE includes a mid-level networking toolkit for times when a high-level, more generic solution is not appropriate. Developers can use this toolkit to quickly craft customized solutions or to build their own reusable, high-level components.

- Low-level networking.

  Some applications have such unique needs that even POE's mid-level tools aren't a good fit. POE can help there as well.

  POE's low-level functions supply just the basic features needed to write effective networked applications. At this level, an application can include entirely custom networking code. Consequently developers have the most control over how that code is written.

  These low-level functions provide a common base for all POE programs. The result is interoperability at every level of abstraction.

2 A Powerful Multitasking Framework

> *[M]any things which cannot be overcome when they are together, yield themselves up when taken little by little.*
>
> — *Plutarch*

POE's multitasking system is based on events and event handlers, rather than forked processes or threads. Applications that use POE for multitasking are therefore portable to environments that don't support forking or threading.

Forked processes and threads are still available when portability isn't an issue, but they are an option rather than a requirement.

2.1 Events & Event Handlers

Events are bits of information that represent interesting occurrences. The most common events represent network I/O, the passage of time, and the status of managed processes. POE will watch for these and others, and it will notify applications when they occur.

Event handlers are routines assigned by an application to be called when events occur. POE ensures that the proper routines are called for each event.

- When there is pending network I/O, POE calls the appropriate handlers to process it.
- When alarms are due, POE wakes up the application by calling the routines that were waiting for them.
- When a managed process does something, POE notifies the application that a result is available.
- And so on.

Event watchers are the parts of POE that detect occurrences and create new events. POE can be extended to watch for new events through a clearly documented interface. This is great for the rare occasions when an application must detect events that POE doesn't already recognize.

2.2 Multitasking with Events and Handlers

Applications often need to wait for many things at once. For example, a network server might watch for I/O on several connections while waiting for a handful of alarms. It may even manage multiple processes in the idle time between network requests.

POE was designed for this sort of simultaneity. It waits for everything an application is interested in, all at once. It notifies the application when things occur, in whatever order they happen. Applications can essentially perform as many tasks as needed, as those needs arise.

2.3 Avoiding the Pitfalls of Forking & Threading

One of the most frustrating aspects of multi-process programming is how difficult it is to share data between them. POE runs all its tasks in a single process, so sharing data between them is as fast and easy as accessing variables.

Threads defeat the memory protections intrinsic to multi-process applications, making it very easy to share data between tasks. Unless used with the utmost of care, however, threads can easily corrupt an application's data and lead to mysterious—and often spectacular—failures.

POE avoids common threading problems by treating its event handlers as "critical sections". Each handler runs uninterrupted, one at a time, until it has completed, intrinsically locking memory access and eliminating the problems that would otherwise occur.

Perl's own threads avoid memory corruption by not sharing data between them by default. Each new thread must therefore build a copy of Perl's interpreter when it's created, and it must destroy that copy when it is done. This causes performance and resource problems in large-scale or dynamically threaded programs. POE's tasks are lightweight by comparison because they share—rather than copy—Perl's interpreter.

2.4 Supporting Forked Processes & Threads

POE also makes it easy to run routines or entire programs in other processes. POE handles the mechanics of spawning these processes and executing code in them, allowing developers to focus on working with their results. Third-party modules like POE::Component::Child make this common task even easier.

Perl's threading is still young and relatively unsupported, in POE and by the Perl community at large. Arthur Bergman, an active POE developer and the author of Perl's threading system, has recently been awarded a grant by The Perl Foundation to add threads support to POE. This will further the development of Perl's threading and bring manageable thread support to POE.

3 A Flexible Message Passing Environment

3.1 Internal Events

Applications based on POE can create ad-hoc events to represent things that happen within themselves. These events are often used to pass messages from one part of an application to another.

POE's message passing is a powerful form of loose coupling that encourages well-defined, modular systems design. This level of modularity facilitates group development by providing definite boundaries between the different parts of an application. Development can easily be performed in parallel once those boundaries—and the interfaces across them—are defined.

For example, an application may have a logging subsystem that accepts and records status messages. Other parts of the application can then send it messages representing their statuses.

3.2 Distributed Programming with External Events

POE's messages can be transparently passed across networks thanks to a free third-party module called POE::Component::IKC. This module adds transparent event passing to POE, letting applications host subsystems on hardware appropriate for their tasks, distribute applications across a farm of machines, or aggregate information from many sources.

For example, a logging subsystem can be placed on a relatively slow machine with a lot of disk storage, while business logic is handled from a relatively fast machine. Or several monitoring nodes throughout an enterprise might ship status events to a centralized data warehouse for analysis.

POE::Component::IKC even includes a lightweight client that provides an interface between POE based applications and clients that don't use POE. It is most commonly used in CGI programs, which must remain nimble yet often need to access powerful, persistent services.

IKC's lightweight client can also be used to transport messages between new POE applications and existing systems. The previous example's monitoring network may already be implemented. Rather than replace a huge network of monitoring nodes, they may be extended with IKC's lightweight client to inject status messages into a POE based aggregator.

True to TMTOWTDI[2], POE::Component::IKC can be reconfigured to transport messages in different formats. If that isn't enough, POE's modular components and mid-level networking toolkit provide everything needed to create entirely new message-passing systems. POE::Component::Server::SOAP was written this way.

[2] "There's more than one way to do it." We're not sure how it's pronounced.

4 A Robust, Tested System

POE was first openly released in August 1998 after almost two years of internal development. POE has evolved since then to match the changing practices and needs of developers.

If you can use Perl, you can use POE. POE is, and has always been, distributed under the same terms as Perl itself. It is entirely open source, and it has benefited enormously from years of peer review.

POE's stability has earned the respect of individuals and companies around the world. Part of this respect comes from POE's extensive test suite. Over 2000 tests barrage it before each release and during most installs. More are added all the time.

As part of our commitment to quality, POE comes with a one-step status reporting facility. If a problem does occur, developers can—at their option—submit the results of POE's tests in their unique environments and situations.

5 A Large and Growing Toolbox

> *All the tools and engines on earth are only extensions of man's limbs and senses.*
>
> — *Ralph Waldo Emerson*

POE includes a large number of tools covering several tasks, and Open Source developers around the world are constantly publishing free, reusable components on the CPAN.

POE continues to become more flexible over time. Currently it includes the tools to make new event watchers. Soon nearly every aspect of POE will be customizable and extensible without directly "hacking" the library itself.

New facilities for adding custom APIs are being developed. The first third-party API is being created right along with them. It will provide deep introspection into POE's innermost workings, allowing advanced developers to debug and tune their applications more effectively than ever before.

6 Success Stories

> *There is a great satisfaction in building good tools for other people to use.*
>
> — *Freeman Dyson*

6.1 Best New Module

The Perl Conference (now OSCON) bestowed upon POE the "Best New Module" award in 1999.

6.2 ActiveState Active Award Nomination

Rocco Caputo was nominated for an ActiveState "Active Award" in 2001 for his work on POE.

> — `http://www.activestate.com/Corporate/Awards/winners.html`

6.3 Need To Know

The perennially hard-to-please folks at "Need To Know" reviewed POE in 2001:

> POE can wrap itself around a GTK loop and manage a fistful of transient, self-contained, self-creating objects. Or it can run a Web-server as a foreground interface to a bunch of background Perl objects doing God knows what. Even NTK's official Perl lamers wrote an HTTP SOAP server which simultaneously controlled an IRC bot by using POE to splice a bunch of CPAN modules together—in an evening. POE is ZOPE without the "Zzzz" of Python. It's small enough to understand, and big enough to extend. It *is* worth a weekend of your copious free time.
>
> — `http://www.ntk.net/2001/05/25/#TRACKING`

6.4 Linux Magazine

The cover of Linux Magazine's October 2002 issue features an article on doing many things at once with POE. In it, Randal L. Schwartz writes:

> POE definitely has some interesting uses, especially for networking code, and it fits well in the "Perl as glue" model.
>
> — `http://www.linux-mag.com/2002-10/perl_01.html`

6.5 Morgan Stanley

At OSCON 2003, Merijn Broeren announced that Morgan Stanley uses POE to monitor over 9000 Unix machines.

6.6 Books

POE will appear in the upcoming new edition of *Perl Cookbook* by O'Reilly & Associates.

O'Reilly will also be publishing an updated *Advanced Perl Programming* containing a chapter on POE.

6.7 Letters

Subject: The future

> [H]aving accidentally been involved in the early days of the SAMBA and KDE development effort, I can tell you that through my own interest I can foresee a very similar and vibrant future for POE. Of all the thousands of projects out there, POE is the only one that has captured my imagination in the last year or so. Guessing from my past good luck/intuition, perhaps it would be wise to start to prepare for the day POE may suddenly become another high-visibility project.
>
> — Mike Wilkinson

Subject: Re: POE::Wheel::Run and SIGPIPE (thanks)

> I just wanted to say thank you for writing such a wonderful framework. You've made life as a Perl programmer so much easier. I just rewrote a piece of software using POE and the code is about $\frac{1}{8}$ the size of the original code. Not only is it much smaller but it's so much easier to debug and extend. The original took about 2 months to write and I was able to rewrite it in about 6 hours using POE. Now that's pretty darn cool.
>
> Thanks for your time and contributions. I really appreciate it.
>
> — Todd Caine

Subject: An extremely geeky post

> Colour me impressed with POE. POE is an event-based object environment for Perl. It provides a framework to handle messaging between different processes, and allocation of processor time—so, for example, it's very good at doing server based things. And the interfaces are designed really well. I managed to hook it into a different event handler, and effectively deal with incoming requests quite happily with my first script, in one attempt. That's after only one evening browsing through the documentation.
>
> — Matt Webb

Subject: Monitoring...

> I currently have POE running with multiple FollowTails on Syslog related log files. Basically, I take in syslog data and translate this log data

into events for our Events display. Right now, I'm running on > 100 different regex patterns and have peaked a bit over 2500 lines / second with [about] 20% CPU... (7 different logfiles...)

— Douglas Stevenson

No subject.

I'm using POE to run our school's Counter-Strike tournament. It's AWESOME. You have no idea how much work it saves me. It configures each server at the beginning of each match, sets up practice times on the servers, sets the server to go live, takes screenshots at the end of the game, and reports scores to the SQL database. Considering we'll have close to a dozen servers and one administrator (me), this will help a LOT.

— A. Chen

Subject: How POE Saved BayCon TV, Helped Godzilla Destroy Tokyo, and Let Me Sleep

Every year in San Jose there's a science fiction convention called BayCon (http://www.baycon.org). A bunch of science fiction fans get together at a DoubleTree hotel for Memorial Day weekend to meet their favorite writers and artists and hang out together. For the people who are actually staying in the hotel, there is an in-hotel TV station (we take over Channel 4) called BayCon TV, or BCTV for short. We play interviews with our guests of honor, footage taped during the con, fan videos, old movies, and slides of science fiction history and trivia. This year, my wife and I were in charge of it.

When BCTV has been running in the past, the crew would build huge videotapes full of content, then have to wake up in the middle of the night to switch them out. If they missed a tape change, or if some footage was shorter than expected, the entire schedule could be thrown off. Finally, they couldn't do things like accept fan videos and digital photos at the convention—everything had been pretaped!

THE PIT AND THE PENDULUM

Since coming aboard as BCTV's general technical person, I'd wanted to try running the station entirely digital. I used Xine, one of the main Linux movie players, as my content display system. I still needed a scheduling system, though—something precise enough to show scheduled movies on time, but flexible enough to deal with unplanned events like missing files. I covered reams of papers with scribbled notes, designing such a system, until I realized that POE could do it all for me.

Thanks to POE, I could turn the entire system into two agents which I ran as sessions: the Player and the Scheduler. Personally, I find it easiest to think of these two as beleagured TV station staffers trying to get some sleep while keeping the station running. Before Scheduler goes to sleep, she checks the listings of upcoming programs and sets an alarm to wake her up right before the next show. When the alarm goes off, she double-checks the schedule and wakes up Player. She tells Player what program needs to be played next, sets her alarm for the next scheduled program,

and goes back to sleep. Player, meanwhile, plays the requested program, then catches some Z's himself.

Substitute some alarm_set() calls for my anthropomorphization above, and you have my playback system. I had originally imagined a huge system using multithreading and frequent time checks. POE let me keep it under 1000 lines of code. We did have some system glitches, but none were remotely related to POE. The playback system was rock-solid—and I hadn't had a chance to give it a full system test before the convention began. I took a gamble that POE would be solid, and I was right. Thanks, Rocco and other POEts.

THINGS TO COME

Our performance this year was good enough that we've been asked to run BCTV next year. I have a host of minor improvements that I'm planning on making, but the biggest feature improvement will be automatic fill generation. Scheduling programs can be a pain, and the hardest part is preventing viewers from having to stare at a blank screen between programs. To that end, we're bringing another agent to our little virtual team. Call him Phil, if you don't mind. Phil's got an overhead projector, a boom box, a few short films, a collection of announcements and slides, and some crayons. His alarm goes off whenever nothing's being played over the TV station, and it's his job to create filler until the next scheduled event.

All of this functionality should be easy with POE—and would be extremely difficult without it. So again, thanks. I'll give updates as the system becomes more mature... and if you're in the San Jose area over next Memorial Day, come to BayCon and look for a (hopefully) relaxed and well-rested man in a Prisoner jacket. That'll be me, and I'll be relaxed and well-rested because of POE.

— Stephen Nelson

7 For More Information

POE's home page

`http://poe.perl.org/`

POE components on the CPAN

`http://search.cpan.org/search?query=POE%3A%3AComponent`

POE's contributed test summaries

`http://eekeek.org/poe-tests/`

POE's CPAN tests

`http://testers.cpan.org/search?request=dist&dist=POE`

POE's mailing list

POE's mailing list is a great place for general questions. For subscription information, send a blank message to <poe-help@perl.org>.

POE's lead developer

Rocco Caputo may be reached by e-mail at <rcaputo@pobox.com>.