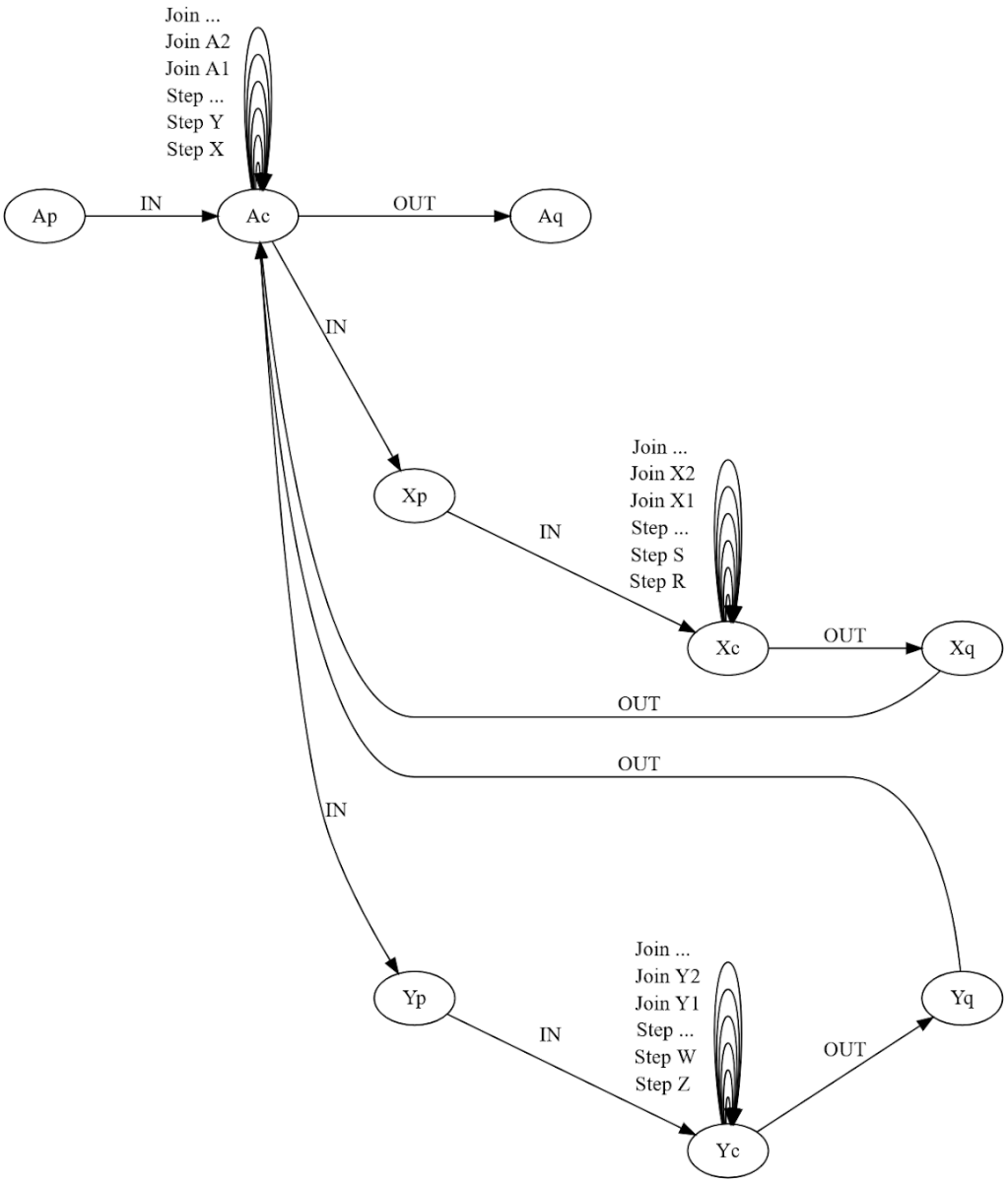
Petra

The future of computer programming, here today.

By CognitionBox.io



Contents

Current Industry Problems

Exponential complexity

Technology gives it's users options. Options to create new products, services, markets and technology itself. This a positive feedback loop, that causes exponential growth in the demand and creation of new technology. As we grow faster and faster we usually forget, ignore or are unaware the technical debt that accumulates during the competitive rush.
To evolve we need to create more powerful layers of abstraction that manage the complexity below, whilst providing the ability to express our intentions easily.

Shortage of skills

As complexity increases there is an ever-increasing relative shortage of skills. We all can't be an expert in everything, our brains have natural cognitive limits, hence we must specialize. This requires intense training and education, which is costly. It's more cost efficient to train someone in one technology that is highly effective at expressing complex systems, than training them in multiple technologies that must be chained together to produce the same result. Not only is the cost of training lower, the resulting architectures are simpler and easier to maintain.

Cost of Testing

The complexity of testing systems in a complete sense, is exponentially proportional to the number of possible system states. This becomes unmanageable for large systems and the problem is magnified when timing can affect the outcome, which is true for any system that requires more than one thing to happen simultaneously. Current software systems consist of multiple applications distributed across multiple machines, communicating via queues, databases and restful web services.
The sheer number of moving parts is enough to blow testing combinations through the roof. We need a way to distribute large systems that reduces the energy needed for testing.

The Solution: Petra

What is Petra?

Petra is a new way to program. It enables a simple way to program high performance complex business logic more formally, thus reducing bugs in complex systems. Petra is a language and runtime specification which can be ported to different environments and technologies. Petra has a reference implementation which has been coded in Java and delegates to Hazelcast for its distributed computing capabilities. Petra and has tools for automatic verification and testing.

What will it be in the future?

Petra aims to be the de facto specification for defining systems and processes in industry. Moving into the future there will be more and more specialized technologies for Petra to target and/or integrate with, spanning in-memory data grids, micro-services, AI, blockchain, FPGAs, neuromorphic, optical and quantum computing.

Benefits of Petra

Less bugs

Petra code is structured in a more formalized manor and provides guarantees about the runtime behaviour of the code. This encourages the programmer to think more clearly about an implementation and to format the code in a more readable / reasonable way. This, combined with automated reachability testing, will dramatically reduce the likelihood of bugs in Java based systems, which will prove invaluable to mission critical applications, such as in finance and defence.

Automated Testing

Petra systems are tested for reachability automatically.
We use machine learning to test all edges and subgraphs within the graph definition, which proves / disproves reachability. This dramatically reduces the software testing effort.

Small Yet Powerful Java DSL

With only a small number of core operations, Petra is quick to learn. From the perspective of a purely object-oriented Java developer, Petra is much easier to learn than other functional programming languages such as Haskell, F#, Clojure and Scala. Those with functional programming experience should learn Petra very quickly thanks to the simple approach taken. Petra allows for creation of complex systems with more concise and clear code. Petra uses a hybrid functional, declarative, disorderly, object-oriented paradigm to scale systems using structured objects and formalize the possible state transitions in a seamless fluid manor.

Java and JVM Compatible

Petra reference implementation is written in Java and thus works effortlessly with existing Java components. Components from Legacy systems can be easily

integrated to a green field Petra system. With Java at its core Petra can be modified to support popular JVM languages, such as Groovy, Scala and Kotlin.

Cross Mode Execution

Petra systems offer the same runtime behaviour across sequential, parallel and distributed execution modes. This helps you to reason, debug, test and scaling your complex business logic.

Parallel and Distributed Runtime optimizations

Petra observes the runtime dynamics of your distributed application and uses machine learning to make intelligent performance optimizations, without affecting the business logic of your application.

Graph and Edge Proxying

Petra allows functionality available only on certain nodes to be targeted automatically without having to directly address the nodes using a URL or IP address. For example, there might be specific nodes that host GPUs, Neuromorphic processors, Optical Processors, Quantum Processors, large multi-core processors, specific sensors, actuators, or sensitive algorithms. This has uses in HPC, Microservices, Robotics and IOT.

Costs of Petra

Petra uses a declarative, disorderly style for programming Java/JVM based systems, a paradigm that requires a little getting used to. There will be a training cost associated with Petra adoption, however the overall cost will be significantly lower compared to adopting other functionally languages.
Although Java compatible, legacy systems will need to be refactored into Petra flows, for maximum benefit. This cost can be justified by the increased reliability and performance provided by Petra.

Petra Implementations

The initial Petra implementation targets Java and Hazelcast IMDG.

Why Java?

We believe Java has a special sweat spot in software engineering. Java has a huge open-source ecosystem and tool set. Java can be used for all sorts of systems and the performance of it's underlying JVM technology continues to improve.

Why Hazelcast?

Hazelcast.com provides distributed versions of Java's core data structures and processors. This means it's easy to create a system that has consistent behaviour across in-process and distributed execution environments.
Hazelcast uses multicast to find nodes within the same network and then automatically clusters these nodes. Deploying a cluster with Hazelcast is quick and easy, and their clusters can scale to hundreds of nodes.

Applications of Petra

Mission critical workflows

Financial Trading, Risk and Defence are both industries that rely on mission critical systems. Any downtime / systematic failures can cause serious consequences. The ability to reason more formally about complex high-performance systems is invaluable in these industries. Petra can help to formalize mission-critical work flows, whilst keeping a high level of performance, maintainability and interoperability with legacy Java systems.

Infrastructure orchestration

With the advent of cloud, infrastructure can be defined and dynamically re-configured using APIs. Petra can be used to formalize your mission-critical cloud infrastructure.

Robotics

The Petra paradigm is well suited to robotic control where data feeds from multiple sensors need to be read and analysed and actions need to be streamed to multiple actuators, all in parallel. This problem is amplified when clustering Robots, i.e. creating robot swarms. Petra can be used to program a whole swarm of networked robots, sharing and combining their computational resources at the same time as targeting specific nodes in the network automatically, without having to address the nodes, all using a simple, functional programming paradigm that can be tested automatically.

IOT (Internet of Things)

The house of the future would contain multiple sensors and would be able to act on the information from all of these in parallel. IOT programming can be viewed as a simpler subset of robotic control and thus Petra would also be very useful in this field.

How Petra works?

Graph Programming

Graphs are currently the state of the art for programming parallel and distributed computations. Graphs are comprised of vertices and edges. Graphs offer endless possibilities, making it harder to reason about correctness. There are currently a handful of Java open-source frameworks that support forms of graph programming, including Apache Spark, Kafka Streams, Flink, Samza etc. These systems offer plenty of flexibility, however for critical environments simplifying the verification process is more important. Also graphs do not work well with traditional programming paradigms such as OOP (Object-oriented programming) which is required for scaling the structure of large and complex systems. Petra's novel approach is to restrict traditional graph programming, whilst being expressive enough to create any sort of system and support OOP. Petra has...

- The ability to nest Graphs inside Graphs, by allowing a graph edge to be substituted for another graph. This provides modularity.

- Graphs with their own pool of states, which is transformed towards an overall desired state.

- Graphs which have their own continuous event loop which applies these transformations until a termination state is reached.

- Graphs which scale automatically, according to the number of states within the states pool that can be operated on in parallel.

- Implementations for the fundamental collections including lists, sets and maps, which can be used in the same way across sequential, parallel and distributed modes.

Petra Features

The Petra innovation is the combination of several features that are synergistic in providing a safe and high performance graph programming model. Each feature helps either the robustness or the scalability of graph programming.

Scalability

Scalability constitutes everything that is required to increase the impact of your software, including scaling up performance and the breath of functionality you need to support. Petra systems can scale in terms of parallelism across cores and machines, and also in terms of breathed and depth of functionality, by providing a simpler and safer programming model for the needs of today.

Automatic Parallel and Distributed execution

Petra works out the order in which operations can execute and dynamically creates computational resources to accommodation varying loads. Efficient fine-grain parallelism is at the Petra core and it's paradigm makes it easy for developers to target the available CPUs.

Modular Graphs

To manage complexity, Petra Graphs can be nested within one another. Each graph has its own type-checked contract which must be compatible with the Graphs it interfaces with, in order for states to flow across.
This means a complex system can be split up into a number of smaller Graphs which can be maintained individually, helping to scale the size of your system.

Cross Mode Execution

Petra systems offer the same runtime behaviour across sequential, parallel and distributed execution modes. This helps you to reason, debug, test and scaling your complex business logic.

Simple deployment

Hazelcast clusters are incredibly easy to deploy, simply run-up nodes within a network and these nodes will auto-cluster using multicast to find one another. Scaling up and down is just a matter of creating new/killing off Hazelcast nodes.

Type-safety & Compiler checked state-flow restrictions

There is a strong type-safety theme used throughout Petra. All graph edges require a compiler checked input/output types, which can be refined with runtime checks. Once the input/output types are declared, the checks and computations are constrained to reference these types, making it difficult to make mistakes. In addition, state domains can be defined which restrict Graphs to operate on specific groups of states.

Automatic Verification

An entire graph can be easily tested though the automatic generation of test data. First a graph is decomposed into its constituent Graphs i.e. all of the subgraphs, down to the edge level. Then each subgraph and edge is tested by generating random data restricted to the required input type, using a combination of reflection, machine learning and heuristics on the source code.

High-performance & Fault-tolerance In-memory computing

Under the hood Petra targets Hazelcast IMDG, which manages the complexities of robust in-memory cluster computing. When deployed with Hazelcast JET (which includes IMDG), Petra can coordinate critical its critical process via IMDG and invoke JET streams for its advanced real-time streaming technology.

The Petra Language

Look and Feel

Below we demonstrate the look and feel of the Petra language with a simple Java like language.

Edges

At its most granular level, Petra executes actual code using codlets defined within an Edge construct. Edges have a generic input type and output type. Edges have a consumes and produces function that corresponds to the pre/post conditions for the edges function. These pre/post conditions use a class type and a predicate to restrict the possible inputs and outputs.

```
class AtoB extends XEdge<A,B> {
     {
          consumes(some(A.class),a->a.isOk());
          function(a -> new B());
          produces(some(B.class),b->b.isOk());
     }
}
```


Extractions

Any class can be marketed with an "Extract" tag meaning that the runtime will recurse any instance of it in order to extract values. This is important when it comes to splitting an object for parallel and distributed processing.

```
@Extract
class X {
     @Extract E e = new E();
     @Extract F f = new F();
     @Extract G g = new G();
}
```

Graphs

Graphs are similar to Edges however they can contain other edges or other graphs. Graphs allow you to create large systems through composition and hiding complexity behind finer grained graphs. Edges are the smallest unit of computation, which can be contained by a graph.

```
class XtoY extends PGraph<X,Y> {
     {
          readConsume(some(X.class),x->x.isOk());
          step(new EtoP());
          step(new FtoQ());
          step(new GtoR());
          join(some(P.class),
               some(Q.class),
               some(R.class),
                    (P,Q,R)->new Y()
               some(Y.class));
          returns(some(Y.class),y->y.isOk());
     }
}
```

Graphs can have zero or more joins which execute sequentially in the order defined, only after all the steps within the graph have finished executing.

Steps can be defined in any order within a graph. Steps will execute as a when their data dependencies have been computed. This is a form of disorderly programming as it doesn't matter which order the steps are listed, all that matters is that the steps work together correctly to produce the right type of value (so in the example above, this would be a value of type Y that returns true when calling y.isOk()).

For more information on Petra please us contact by:

email: hello@cognitionbox.io
tel: +44(0)20 3290 3708

